THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Aspects on asset liability management via stochastic programming

Fredrik Altenstedt

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematics
Chalmers University of Technology and Göteborg University
Göteborg, Sweden 2003

Aspects on asset liability management via stochastic programming
Fredrik Altenstedt
ISBN 91-7291-342-8

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 2024
ISSN 0346-718X

Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 772 1000

Printed in Göteborg, Sweden 2003

Aspects on asset liability management via stochastic programming

Fredrik Altenstedt

Department of Mathematics
Chalmers University of Technology
Göteborg University

Abstract

This thesis consists of four papers, all considering different aspects of asset liability management (ALM) modelled by stochastic programming. The main motivation for the work was research performed for a partner company offering life insurance policies.

In the first paper, we present our model of a Swedish life insurance company. The model is based on stochastic linear programming. We describe the different parts that such a model needs, such as a model of the company itself, and a model of the surrounding economy, including a model of the customers' behavior.

Paper II develops methods for speeding up the nested Benders algorithm for multi-stage stochastic linear programming, when applied to medium-sized problems. We show how time-savings on the order of 25% may be achieved by dividing the scenario tree into blocks and altering the sequencing protocol to take advantage of this block-structure.

When developing the model in paper I, our partner company wished to use 7 asset-classes in the model. This desire requires us to have rather broad scenario trees, if the solution is not to chase spurious profits present only in the scenario tree. In paper III we show that by reducing trade to much fewer linear combinations of asset classes, we may improve the quality of the solution, as long as the linear combinations are carefully chosen. We illustrate this on a small test-problem, where the linear combinations are found using dual information from the ALM problem.

In papers I–III, the modeling paradigm chosen has been stochastic linear programming. However, this framework rapidly becomes computationally infeasible for problems with many time-stages. In paper I this has been solved by aggregating time-stages, in effect reducing the feasible set to a singleton at some of the time-stages. Another possible path is the use of parameterized policies. In paper IV we compare stochastic linear programming to a parameterized dynamic policy, both quantitatively and qualitatively. Furthermore, we construct a hybrid policy, where solutions from a stochastic linear programming approach are used to create a parameterized function. We show that an optimized such policy significantly increases the performance compared to pure stochastic linear programming.

Keywords: optimization; stochastic programming; asset liability management; life insurance

This thesis consists of an introduction and the following papers:

Paper I F. Altenstedt, *An asset-liability management system for a Swedish life insurance company*, submitted to Annals of Operations Research.

Paper II F. Altenstedt, *Memory consumption versus computational time in nested Benders decomposition for stochastic linear programming*, submitted to Computational Optimization and Applications.

Paper III F. Altenstedt, *Asset aggregation in stochastic programming models for asset liability management.*

Paper IV F. Altenstedt and M. Patriksson *Policy optimization: parameterized decision rules vs. stochastic programming for asset liability management.*

Contents

Acknowledgments

First and foremost, I would like to thank my supervisor, Michael Patriksson. His continuous encouragement and perpetual enthusiasm have helped me enormously throughout this work.

I would like to thank Ann-Brith Strömberg, both for checking parts of this manuscript and for our smooth cooperation while teaching.

I am grateful towards Nordea life & pension who have contributed to this thesis, with monetary support, data and access to the inner workings of a life insurance company.

Furthermore, I would like to thank (in order of appearance)..

My sister and parents, no motivation necessary.

My friends, for distracting me, occasionally against my vivid objections.

My colleagues and fellow PhD student, for making MC a nicer place to work. Our conversations over lunch have always been entertaining, although sometimes surreal.

And finally, thanks to Lena, for everything.

Fredrik Altenstedt
Göteborg, September 2003

1 Introduction

The primary objective of the work leading to the thesis was to develop tools to help a Swedish life insurance company make better decisions. Such a company must make two major decisions: how to invest the assets and how to set the bonus rate of return, meaning that a large part of their decision is a portfolio selection problem.

1.1 Background to Swedish life insurance

The main product offered by our partner company is what is known as a traditional pension insurance policy. Under such a policy, the company guarantees that the customer will receive a monthly payment from the age of 65, continuing for a number of years stipulated in the policy. In exchange for this promise, the customer pays to the company a sum of money, which will be invested until the customer retires. If the company makes successful investments, the funds will more than suffice to cover the promises to the customers. In case this happens, according to Swedish rules for mutual companies, all excess profits are distributed to the customers as a bonus return. One peculiarity of the Swedish system is the fact that the bonus rate already allocated to the customers may be reduced retroactively if the company becomes insolvent.

Since the guaranteed monthly payment is rather low, most customers expect the bonus payments to provide a substantial fraction of the future retirement pay-outs. Hence the task of the company is to invest the means obtained from the customers in a way that will give a good return on the customers' payments at an acceptable risk, while making sure that the return on the assets is at least sufficient to cover the guaranteed payments promised in the policies, preferably without ever having to reduce the bonus allocations retroactively.

In order to guarantee that the company is on track towards covering its promises to the customers, the regulating authorities require the company to abide by two sets of rules. The first makes sure that the company covers the guaranteed payments, while the second is designed to make sure that the company is sufficiently solvent to honor the bonuses distributed to the customers. Both these sets of regulations take the form of reserves, which need to be covered by the assets of the company. The overall decision-problem therefore is an *asset liability management* (ALM) problem. The size of the first reserve is loosely coupled to the prevailing long-term interest rates, while the other is determined by the bonus rate of return given. Hence, the company need to consider the correlation between the asset yields and the long-term interest rates when determining the portfolio, in order to make sure that the first reserve is covered. In the same fashion, the setting of the bonus rates and the investment portfolio selected needs to be coordinated in order to lower the risk of the company being forced to retroactively lower the bonus allocations.

1.2 Portfolio optimization

A good investment portfolio is one which gives a high expected yield at low risk, with risk defined by an appropriate risk measure. When Markovitz [34] introduced his mean-variance portfolio selection technique, the risk measure chosen was the standard deviation of the realized return. Another approach to model the risk-adverse behavior of a rational investor, integrating risk and reward, is the use of utility functions, as described by Von Neumann and Morgenstern [46]. Other risk measures beside variance includes value-at-risk (see J.P. Morgan [29]), conditional value-at-risk, as used by Andersson, Mausser, Rosen and Uryasev [1], and absolute deviation, used by Konno and Yamazaki [31]. For all these problems, only one decision is made, and the portfolio is considered to be fixed over the decision horizon.

In multi-stage portfolio problems, the portfolio is not considered to be fixed over the investment horizon, but instead periodically rebalanced. Hence, the optimal decision is no longer a portfolio, but a strategy, describing how the portfolio should be balanced for each period in time. Both the mean variance and utility function frameworks have been extended to the multi period case by Mossin [23] and Hakansson [23, 24].

In the articles cited above, the problems are pure policy optimization problems since no special purpose of the yields of the investment is specified. When running an insurance company, the purpose of the investments is to be able to cover the payments to the policy holders. Hence it is more appropriate to formulate the decision problem as an asset liability problem (ALM), where the liabilities are explicitly considered when the portfolio is chosen. Again, ALM problems may be single-stage or multi-stage problems. The first multi-stage models formulated, such as by Chambers and Charnes [14], assume the future to be perfectly known. More advanced models, such as the one formulated by Bradley and Crane [9], incorporate the randomness inherent in portfolio selection. When randomness is involved, we must determine what it means for a solution to be feasible. We may either require the constraints to hold with probability 1, or require the constraints to hold with a predefined probability, resulting in a chance constrained problem (see Charnes and Cooper [15]). This kind of constraints are included in Dert's [20] model of a Dutch pension fund.

One type of life insurance product offered by some life insurance companies (although not by our partner) is an equity linked contract with guarantee. Such a policy will pay the maximum of a guaranteed amount and the value of an index when the contract matures (i.e., the policy-holder reaches a certain age, or dies). Brennan and Schwarts [10] show how these policies may be decomposed into the guaranteed amount and an option with the strike-price given by the guaranteed amount. This decomposition is the basis of an investment strategy where the company tries to replicate the option. Furthermore, Brennan and Schwarts suggest that different investment strategies themselves may be considered as assets in the portfolio selection problem.

However, option replication does not appear to be suitable for the Swedish conditions, since there is no clear rule of how much the policy holder receives at maturity. (The reason for this is that the company itself sets the bonus rate of return.) Instead, we choose to model our problem as a multi-stage stochastic linear program. In the literature these exist several models of this type dealing with ALM, the most important ones being Kusy and Ziemba [32], Cariño et al. [12, 11, 13], Dert [20] (modified by Gondzio and Kouwenberg to remove the chance constraints [22]), Consigli and Dempster [17] and Høyland and Wallace [28].

1.3 Multi-stage stochastic programming

If we limit the number of points in time where decisions can be made to a finite set, and assume that we do not wish to have any chance-constraints in our model, then the ALM problem may be formulated as that to:

$$
\begin{aligned}
\underset{x_t, t\in\{0,\dots,T\}}{\text{minimize}} \quad & f_0(x_0(\xi_0),\xi_0) + \mathsf{E}^P_{\xi_1}[f_1(x_1(\xi_1),\xi_1) + \mathsf{E}^P_{\xi_2|\xi_1}[f_2(x_2(\xi_2),\xi_2) \\
& + \dots + \mathsf{E}^P_{\xi_T|\xi_{T-1}}[f_T(x_T(\xi_T),\xi_T)]\dots]], && \text{(1.1a)}\\
\text{subject to} \quad & x_0(\xi_0) \in X_0(\xi_0), && \text{(1.1b)}\\
& x_t(\xi_t) \in X_t(\overrightarrow{x}_{t-1}(\xi_t),\xi_t), \quad t = 1,\dots,T. && \text{(1.1c)}
\end{aligned}
$$

Here, ξ_t represents all random information known at time t, defined on a standard probability space $(\Omega, \mathscr{F}, P)$, and E^P means that expectations are taken using this measure. All possible random outcomes of ξ_t define a filtration $\mathscr{F}_t$ on this probability space, where $\mathscr{F}_t$ will consist of all events whose outcome is known at time t. The decision variables x_t are said to be fulfilling *non-anticipativity* if they only depend on information available at time t, which is equivalent to require that x_t is measurable with respect to $\mathscr{F}_t$. We use the shorthand notation $\overrightarrow{x}_t(\xi_t) = (x_0(\xi_0), x_1(\xi_1), \dots, x_t(\xi_t))$ to represent the entire decision history. In each period, the decisions made must be feasible, and the set of feasible decisions is influenced both by random outcomes and previous decisions, as indicated by (1.1c).

Since the variables $x_t(\xi_t), t > 0$, depend on random outcomes, the space of possible decisions will be infinite-dimensional if the space of possible outcomes Ω is not finite. Most computational methods which may be employed to solve (1.1) assume that the space of possible random outcomes is finite, meaning that if this is not true, the outcomes need to be discretized into a scenario tree. If the problem fulfills additional requirements such as iso-elastic utility (ruling out most, if not all ALM models), no serial correlations in the random variables, and an objective function which separates over the time-stages, then dynamic programming may be used for problems with continuous states (see for instance Samuelson [40]). Dynamic programming may also be used for low-dimensional state spaces and separable objective functions, but this type of approach is rare for ALM problems as portfolio allocations tend to be continuous. Other problems are more suited for this kind of approach, such as the stochastic unit commitment problem (Takriti, Krasenbrink and Wu [43]).

The most common approaches to the problem (1.1) are decomposition techniques. When decomposing the problem into scenarios, one problem is formulated for each possible random outcome and the non-anticipativity of the problem is enforced using explicit constraints. The non-anticipativity constraints are relaxed using Lagrangian relaxation, and the resulting dual problem is solved using dual ascent methods. A similar method is the progressive hedging method by Rockafellar and Wets [38], which uses the previous iterations' average value of x_t over each information set (the subset of Ω over which ξ_t is constant) to construct the relaxation.

The problem (1.1) may alternatively be decomposed by time-stages, resulting in the L-shaped decomposition method by Van Slyke and Wets [45], which was extended to multiple stages by Birge [4]. As with most LP-based decomposition methods, L-shaped decomposition may suffer from slow convergence, as the solution sequence is unstable. The proposal by Ruszczyński [39] is to introduce a regularization cost-term which stabilizes the sequence of solutions. This has the added advantage of making it possible to delete inactive cuts while still guaranteeing convergence.

Instead of utilizing the structure of the problem (1.1) to decompose it into different subproblems, it is possible to utilize the structure of the problem while solving the deterministic equivalent directly. For example, Birge and Qi [7] utilize the constraint matrix structure when devising an interior point method for a two-stage problem. Further progress for multi-stage problems has been made in this area by Steinbach [42] and Blomwall and Lindberg [8].

1.4 Thesis objectives

The main theme of the work leading to this thesis is to create an ALM system for a Swedish life insurance company. In order to work properly, such a system needs a number of features:

- We need a model of the *company* which includes all relevant constraints on the problem, such as trading restrictions, statutory rules and so forth.
- We need a model of the company's *liabilities*, and how these liabilities are affected by the decisions of the company and the random outcomes of the economy model.
- We need a model of the *surrounding economy*, capable of producing a scenario-representation of the unknown random parameters which are consistent with the subjective beliefs of the decision makers.
- We need *algorithms* capable of solving instances of the problem of relevant sizes.
- We need to be able to interpret the output of the model, in order to turn the output into advice to the decision-makers.

Paper I in this thesis is intended to address the three first items in this list. In it we develop an asset liability model for a Swedish life insurance company. The purpose of this model was to serve as a prototype, demonstrating the advantages of dynamic asset liability management over the company's current static approach, and hence to motivate the company's effort to develop a more realistic asset model. Unfortunately, the negative development of the Swedish stock market during the project's lifetime proved the benefits of using ALM in a more effective way: the losses incurred by our partner company required the owners to lend the company substantial amounts of money, in order to guarantee their solvency. These loans were given on the condition that the company temporarily adopted a bonds only investment strategy. Since a major purpose of our ALM model is to determine how large a fraction of the available assets should be invested in stock, further developments and implementation of the model have been put on hold, possibly indefinitely. Furthermore, the company has been placed in run-off mode and does not accept any new customers.

However, the experiences from the modeling led to some other insights. Testing an ALM model requires solving many stochastic programming problems of the same shape, differing only in the random parameters. **Paper II** is devoted to increasing the speed of solving stochastic linear programming problems of medium size (on the order of 2e5–1e7 variables) on normal office computers, as we have access to a large number of PCs during the night.

Our partner company wished to use their current asset classification, having 7 different asset classes (although 2 classes were not actively traded). Describing the joint probability function of so many assets to a satisfactory degree requires the scenario tree to have a large number of branches, making the resulting problem hard to solve. If we have too few branches to accurately describe the probability function, the optimization will seek to exploit spurious investment opportunities which look good when evaluated over the scenario tree, but bad when evaluated under the true probability measure. In **Paper III** we try to constrain the feasible set of our ALM problem, in order to limit the possibility to chase spurious profits, while at the same time not reducing the performance of the model by still retaining a near-optimal solution inside the feasible set.

A stochastic programming based decision support system may be seen as a black box by the users. A user inputs his/her subjective view of the probability distribution of future events and risk tolerance as well as the state of the company, pushes a button and out pops a decision. For a user, it is not clear how this decision was obtained, and how alterations to the input will affect the output. In an attempt to visualize how the optimal solution from the stochastic programming model would vary with input, we solved a large number of problems and stored the results. By interpolating between the results stored, we may draw graphs of approximately how the decision changes with the state of the company. We then realized that these tables may be used as a proxy for obtaining decisions from the stochastic programming model.

The obvious drawback of this approach is that we no longer get an optimal decision for a given state of the company. There are however a number of advantages.

Firstly, the decision for each possible state of the world is now explicitly available and may be inspected for scrutiny. Furthermore, simulating using the system becomes significantly easier; simulating hundreds of scenarios now just takes a few seconds, while simulating the same scenarios using the full stochastic programming formulation takes hours, if not days. Secondly, the resulting policy may be manually altered, to see how changes in the policy will affect the distribution of key values at the end of the test scenarios. Hence, the construction of these table-based policies is a contribution towards addressing point 5 in the list above. In **Paper IV** we compare stochastic programming to table-based policies as well as another dynamic policy by using a simple ALM problem.

1.5 Contributions of this thesis

1.5.1 Paper I

The main contribution of this paper is the formulation of the model. A property of the model not found in other similar models is the strong connection between the company's actions and the size of the liabilities. In addition, one part of the numerical tests compare different shapes of the scenario tree. Trees with approximately the same number of scenarios are compared, but the trees have different numbers of stages, and a different number of branches at each stage. The different shapes are created by aggregating time-stages in the model. Since the size of tree-based stochastic programming models are exponential in the number of time-stages, aggregation of time-stages is a necessity in order to make the problem computationally tractable, if the number of time-stages is large. Stages are aggregated in this fashion in the models described by Cariño et al. [12], Hilli et al. [25] and Høyland et al. [27], although results from different ways of aggregating the time-stages are not reported for any of these cases.

The previous tests on multi-stage ALM-models we are aware of have either used test scenarios with few enough stages to make stage aggregation unnecessary, or have not tried different shapes of the tree. In our simulations, we find that the longer trees perform better, even when the number of branches are too few to give a good description of the possible random outcomes, as evidenced by a very poor solution stability between different scenario trees. This seems to imply that aggregating time-stages gives a significant bias to the solution.

1.5.2 Paper II

In recent papers on implementations of the nested Benders algorithm the main focus has been on making larger and larger problems solvable on bigger and bigger computers (see for example Dempster and Thompson [18], Birge et al. [5] and Gondzio and Kouwenberg [22]). In this paper we instead focus on trying to improve the algorithms efficiency for medium sized problems, that is, problems of approximately the size which need to be solved when an ALM system is tested by rolling horizon simulations. We find that by tuning the algorithm to make use of most of the available memory, some 15–30% of the computational time may be saved.

This tuning is performed by combining depth-first search of the scenario tree with changes to Wittrock's [47] fast-back–fast-forward sequencing protocol commonly used for nested Benders decomposition.

1.5.3 Paper III

This paper deals with the aggregation of assets into asset classes in stochastic programming ALM models. We show how the selection of the number of assets to include in an optimization model affects the quality of the solution in two ways when an approximation of the probability measure is used. Firstly, the quality of the solution is affected negatively, as we may make all optimal or near-optimal solutions infeasible. Secondly, on the positive side, the aggregation prevents the exploitation of spurious profits emanating from a poor approximation of the full probability measure. Furthermore, we show how dual information from the scenario tree may be used to guide the asset aggregation procedure.

1.5.4 Paper IV

In this paper we directly compare stochastic linear programming to a parameterized *dynamic* policy. Previous comparisons in the literature have been with re-optimized fixed-mix policies. Furthermore, we construct a hybrid policy, where a parameterized policy is constructed from a number of stochastic linear programming solutions. We show that by optimizing the parameters of this policy we significantly outperform the solutions from a pure stochastic linear programming formulation.

1.5.5 Software

In addition to the papers I–IV, the thesis work has resulted in two programs which we hope are useful for other stochastic programming researchers and practitioners. The first program, BNBS, is an efficient implementation of the nested Benders algorithm (the implementation is described in Paper II). The second program, not described in any paper but used in Papers I, III and IV, is an extension of an algebraic modeling language, which allows it to handle stochastic programming problems. This program, STOCHPLAM, is described in Section 3 below. Both programs are freely available to the research community.

2 Summary of papers

2.1 Paper I

An asset liability management system for a Swedish life insurance company

This paper presents our model of a Swedish life insurance company. In order to put our model into perspective we give an overview of other models used for asset liability management for pension funds and life insurance, as well as an overview of the sets of rules under which these models are designed to operate.

The model itself consists mainly of two parts: a model of the surrounding economy (including the customers), and a model of the company, which both are described in the article. Furthermore, we perform a set of numerical tests; we compare the performance of our model to that of a fix-mix benchmark, and we try to determine the effects of aggregating stages in the scenario tree. We show that significant performance is gained if we use trees with many stages and few branches per stage, compared to using shorter trees with more branches per stage.

2.2 Paper II

Memory consumption versus computational time in nested Benders decomposition for stochastic linear programming

Testing a stochastic programming based ALM system is usually done by rolling horizon simulations on a number of test scenarios. Performing rolling horizon simulations involves solving a large number of problems, with slightly different random data. Preferably, these test problems should be solved by utilizing unused resources, such as by borrowing PCs used for normal office work whenever they are not used for their intended purpose. This means that the hardware we use will be a collection of heterogeneous computers connected through a normal office network. Furthermore, a computer may become unavailable without prior notice whenever it is needed for other tasks. These conditions imply that it is easier to parallelize tests by scenarios (i.e., run one scenario per computer) instead of parallelizing the solver (i.e., use all computers to solve for one scenario quicker).

In the literature there exist several implementations of the nested Benders algorithm for solving stochastic programming problem. Many of these implementations focus on making larger and larger problems solvable in shorter and shorter time, on bigger and bigger computers. Less effort has however been spent on trying to shorten the solution times for solvers running on smaller computers with limited memory.

In this paper we explore a number of strategies which may be used in order to trade lower memory consumption for higher computational time. One factor greatly affecting how much memory is consumed is the number of subproblems kept in memory. Traditionally, one subproblem per node in the scenario tree is kept if the problem is small, while one problem per period is kept if the problem is slightly larger. If the tree is searched breadth-first, then these are in fact our only options. If we instead perform a depth-first search, then we also have the option to keep an entire subtree in memory, while sharing LP-structures between the subproblems (see Figure 1). This is done by cutting the problem into two halves by the use of a block-stage. One LP is kept per node before the stage, while the subtrees after the block-stage share LPs.

We show how keeping subtrees in memory may reduce the overall computational time, without leading to much higher memory requirements. Doing this requires us to modify the fast-back–fast-forward sequencing protocol (as given by Wittrock

[47]), which is recommended for stochastic programs both by Gassman [21] and Birge et al. [5].

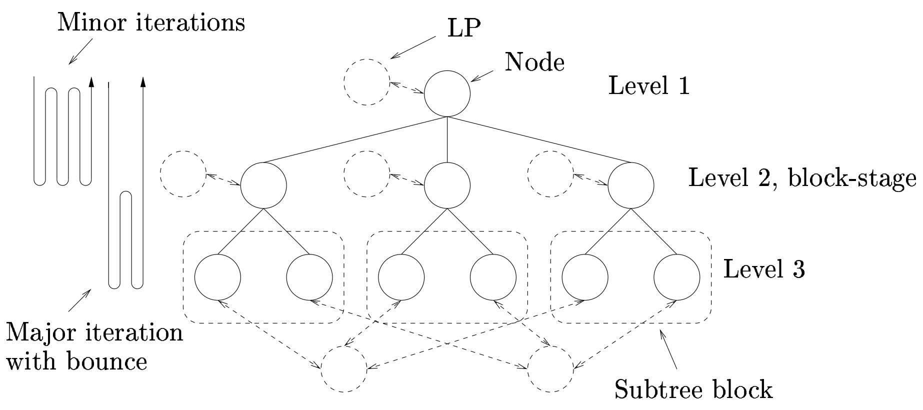

Figure 1: Sharing of LPs.

The reductions in solution times vary between different test-problems; we save between 15–30% of the computation time compared to using one problem per level in the tree, sometimes while also saving memory, but usually by approximately doubling the memory requirements.

2.3 Paper III

Asset aggregation in stochastic programming models for asset liability management

If we look at the problem (1.1), we see that we may define its value recursively as

$$\begin{aligned} g_T(\overrightarrow{x}_{T-1}, \xi_T, X, P) &:= \min_{x_T \in X_T(\overrightarrow{x}_{T-1}, \xi_T)} f_T(\overrightarrow{x}_{T-1}, x_T, \xi_T), && \text{(2.1a)} \\ g_t(\overrightarrow{x}_{t-1}, \xi_t, X, P) &:= \min_{x_t \in X_t(\overrightarrow{x}_{t-1}, \xi_t)} (f_t(\overrightarrow{x}_{t-1}, x_t, \xi_t) \\ &\quad + \mathsf{E}^P_{\xi_{t+1}|\xi_t}[g_{t+1}(\overrightarrow{x}_{t-1}, x_t, \xi_{t+1}, X, P)]). && \text{(2.1b)} \end{aligned}$$

Clearly, the function g_t is dependent both on the feasible set X and the probability measure P used to compute the expected averages. We may thus formulate the problem (1.1) as that to

$$\underset{x_0 \in X_0}{\text{minimize}} \quad f_0(x_0(\xi_0)) + \mathsf{E}^P_{\xi_1}[g_1(x_0, \xi_1, X, P)]. \tag{2.2}$$

However, as stated previously, we are not able to address this problem directly. Instead we discretize the space of possible outcomes into a scenario tree, and solve the approximate problem

$$\underset{x_0 \in X_0}{\text{minimize}} \quad f_0(x_0(\xi_0)) + \mathsf{E}^{\bar{P}}_{\xi_1}[g_1(x_0, \xi_1, X, \bar{P})], \tag{2.3}$$

where $\bar{P}$ denotes the probability measure corresponding to the scenario tree. In this problem, even if the probability measure $\bar{P}$ provides an accurate description of the full probability measure P for the first-stage random outcomes, the resulting optimal first-stage solution may be far from the real optimal solution, if $g_1(x_0, \xi_1, X, \bar{P})$ is a poor approximation of $g_1(x_0, \xi_1, X, P)$. For example, if we have many assets to invest in, but few branches in P at later stages in the tree, there will exist spurious profits in the scenario tree at later stages, and hence $g_1(x_0, \xi_1, X, \bar{P})$ will be an over-estimation of $g_1(x_0, \xi_1, X, P)$. In this paper we experiment with restrictions on the feasible set (replacing the feasible set X by $\bar{X} \subset X$). The purpose is to make $g_1(x_0, \xi_1, \bar{X}, \bar{P})$ a better approximation of $g_1(x_0, \xi_1, X, P)$ than $g_1(x_0, \xi_1, X, \bar{P})$, which will improve the first-stage solution.

When an ALM problem is specified, an important choice is which asset classes to include. Typically, the assets are aggregated into categories such as foreign stock, domestic bonds, etc. If too few asset classes are used, then the model may perform poorly since the feasible set does not contain any good solutions. On the other hand, including too many assets leads to computational difficulties: as the joint probability distribution of the possible asset yields are represented using scenarios, increasing the number of assets requires us to increase the width of the scenario trees, thus making the problem much larger. In this paper, the restrictions of the feasible sets discussed above takes the shape of aggregating assets into bigger asset classes (referred to as synthetic assets). The aggregation is guided by dual information from the ALM-problem, in order to find restrictions on the feasible sets which removes the tendency to chase spurious profits, while at the same time permits the solver to find good asset combinations. We try this approach on a bare-bones ALM problem, and find that there is no significant advantage of using more than two asset combinations, as long as they are carefully chosen. We further find that significant performance is lost if the liability side is not considered when the assets are aggregated into asset classes, either by using dual information from the ALM, or by aggregating assets using Markowitz method, augmented with the liability hedging credit of Sharpe and Tint [41].

2.4 Paper IV

Policy optimization: parameterized decision rules vs. stochastic programming for asset liability management:

The concept of stochastic programming over a scenario trees is only one of many possible approaches that may be used for asset liability management. Another possible approach is the use of parameterized policies (see Mulvey [35]).

If we look at the problem (1.1), and assume that the variables may be divided into one set representing the state of the company, x, and one set representing the decision we may make, y, then the problem may be reformulated as:

$$\begin{aligned}\min_{y_0} \quad & f_0(x_0) + \mathsf{E}_{\xi_1} \min_{y_1(\xi_1)} [f_1(\overrightarrow{x}_1(\xi_1), \xi_1) + \mathsf{E}_{\xi_2|\xi_1} [\min_{y_2(\xi_2)} f_2(\overrightarrow{x}_2(\xi_2), \xi_2) \\ & + \cdots + \mathsf{E}_{\xi_T|\xi_{T-1}} [\min_{y_T(\xi_T)} f_T(\overrightarrow{x}_T(\xi_T), \xi_T)] \cdots]], \qquad (2.4a)\end{aligned}$$

$$
\begin{array}{lll}
\text{s. t.} & x_0 = g(\bar{x}, y_0), & \qquad (2.4b) \\
& y_0 \in Y_0, & \qquad (2.4c) \\
& x_t(\xi_t) = g(x_{t-1}(\xi_{t-1}), y_t(\xi_t), \xi_t), \qquad t = 1, \dots, T, & \qquad (2.4d) \\
& y_t(\xi_t) \in Y_t(x_{t-1}(\xi_{t-1}), \xi_t), \qquad t = 1, \dots, T. & \qquad (2.4e)
\end{array}
$$

In this formulation, the state of the company at time t is a direct function of the state of the company at time $t-1$ and the decision made at time t, as expressed by $x_t(\xi_t) = g(x_{t-1}(\xi_{t-1}), y_t(\xi_t), \xi_t)$. As has been mentioned earlier, the dimension of a stochastic programming problem is high, as one decision may be made for each possible random outcome. Applying parameterized policies is an attempt to reduce the dimension of the problem, by making the action taken at time t, y_t into an explicit function of the state of the company at time $t-1$ and the random development between $t-1$ and t. This explicit function is made dependent on a number of parameters α,which we optimize over, giving us the problem formulation:

$$
\begin{array}{lll}
\min\limits_{\alpha} & f_0(x_0) + \mathsf{E}_{\xi_1}[f_1(\overrightarrow{x}_1(\xi_1), \xi_1) + \mathsf{E}_{\xi_2|\xi_1}[f_2(\overrightarrow{x}_2(\xi_2), \xi_2) + & \\
& \cdots + \mathsf{E}_{\xi_T|\xi_{T-1}}[f_T(\overrightarrow{x}_T(\xi_T), \xi_T)] \cdots]], & \qquad (2.5a) \\
\text{s.t.} & x_0 = g(\bar{x}, y_0), & \qquad (2.5b) \\
& y_0 \in Y_0, & \qquad (2.5c) \\
& x_t(\xi_t) = g(x_{t-1}(\xi_{t-1}), y_t(\xi_t), \xi_t), \qquad t = 1, \dots, T, & \qquad (2.5d) \\
& y_t(\xi_t) \in Y_t(x_{t-1}(\xi_{t-1}), \xi_t), \qquad t = 1, \dots, T, & \qquad (2.5e) \\
& y_t(\xi_t) = \gamma(x_{t-1}(\xi_{t-1}), \xi_t, t, \alpha), \qquad t = 1, \dots, T. & \qquad (2.5f)
\end{array}
$$

The characteristics of this problem is rather different from (2.4); that problem is large-scale but linear, whereas (2.5) is low-dimensional but non-linear.

In **Paper IV** we compare stochastic programming to parameterized policies, both quantitatively and qualitatively. The main advantage of a stochastic linear programming approach is its great flexibility (complex constraints may be handled in a straightforward way), and the fact that we do not influence the solution with prior assumptions of what characterizes a good solution. The major downside is the computational price: the number of variables grows exponentially with the number of time-stages in the problem, forcing us to aggregate time-stages for problems with a high number of stages. On the other hand, parameterized problems grow only linearly in the number of time-stages, and produce problems of low dimension compared to stochastic linear programming. The downside of ALM-problems based on parameterized policies is that they result in nonlinear and non-convex problems which are hard to solve. In order to benefit from the advantages of both methods, combine the two approaches into a hybrid method, by constructing a parameterized policy where the policy function is based on the solution of a large number of stochastic linear programming problems. In our test the hybrid policy clearly outperforms a stochastic linear programming problem, showing us that the aggregation of stages causes a bias which negatively affects the solution, and that a part of this bias may be removed by using the hybrid policy.

3 STOCHPLAM

The formulation of a mathematical program is greatly simplified by the use of an algebraic modeling language (AML). Not many AMLs are however suited for stochastic programming. When the work on this thesis was started, there existed (to our knowledge) only two systems for managing stochastic programming models: SLP-IOR by Kall and Mayer [30], and STOCHGEN, created by Poiré [37] and used by Consigli and Dempster [17]. In addition, there existed a system called SETSTOCH created by Condevaux-Lanloy and Fragniere [16], designed to extract the structure from a stochastic programming problem formulated in GAMS. Since then, other implementations have been created, such as SPInE, described by Valente, Mitra, Poojari and Kyriakis [44], and an extension to the AMPL modeling language, made by Lopes [33]. All these systems, except SETSTOCH, works as wrappers around an existing modeling language; the modeling language is used to create one MPS-file (a standard file format for linear mathematical programs) for each scenario (or node, for some systems) in a scenario tree. These files are then processed to form a stochastic programming problem in SMPS format [3]. Implementing a stochastic programming modeling system in this fashion is practical, since no changes need to be made to the modeling language. However, there are some disadvantages; as the AML is invoked for each scenario or node in the scenario tree, non-random elements of the problem will be processed multiple times, which is inefficient. Furthermore, what is communicated to the solver are realizations of coefficients in the constraint matrix, dependent on the random parameters of the problem, and not the random parameters themselves. Since the solver does not have any information on how the random elements in the constraint matrix are formed from the random parameters, it becomes harder to implement sampling based solution schemes (such as EVPI-based sampling, Dempster and Thompson [19]). (Either the solver has to have information on the joint distribution of the matrix elements, making the implementation model-specific, or the AML has to be invoked whenever sampling takes place.)

We have modified an existing open source mathematical programming language, PLAM, implemented by Barth and Bockmayr [2], in order to be able to formulate linear stochastic programming problems. The alterations make it possible to declare some parameters as random, so that when the model is translated to a constraint matrix, the matrix elements will consist not of numbers, but of expressions containing random parameters. Hence, what is exported from STOCHPLAM is a normal stochastic linear programming formulation:

$$\begin{aligned}
\text{minimize} \quad & \sum_{t=0}^{T} c_t^{\mathrm{T}} x_t, \\
\text{subject to} \quad & W_0 x_0 = h_0, \\
& \sum_{k=0}^{t-1} A_{t,k} x_k + W_t x_t = h_t, \quad t = 1, \ldots, T, \\
& x_t \geq 0, \quad t = 0, \ldots, T,
\end{aligned}$$

where some matrix elements are given as expressions of random variables. These expressions are fed to an evaluation routine inside the solver in order to formulate the full stochastic linear programming problem. The random information is thereby added to the problem inside the solver, not inside the modeling language as with other languages, a difference which is illustrated in Figure 2. With our setup, importance sampling is easier to implement and any redundant evaluation of non-random elements is avoided. A disadvantage is naturally that the solver must include the functionality to evaluate expressions, given the values of the random parameters. This disadvantage is not great, as expression evaluation routines are freely available. When using a wrapper around an existing AML, the already existing expression evaluation functionality of the AML is used, making the solver less complex.

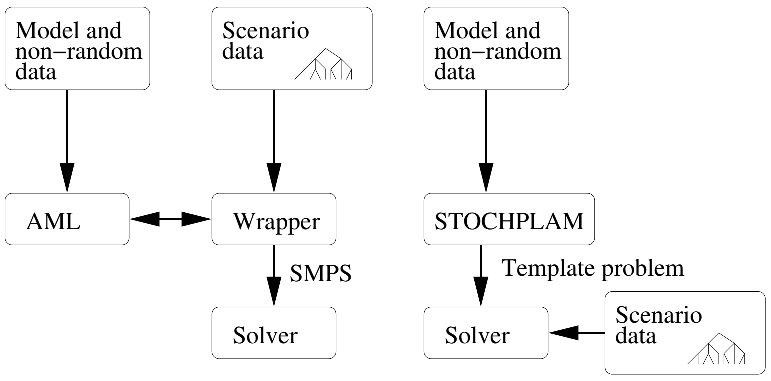

Figure 2: Comparison between STOCHPLAM and other SP modeling systems.

Currently, the only solver supporting output from STOCHPLAM is BNBS (the solver used in Paper II). However, as BNBS may be set to export SMPS files without solving the problem, it is possible to use STOCHPLAM in conjunction with other stochastic programming solvers as well, by using BNBS as a translation layer.

4 Future work

Apart from the ideas for future work mentioned in papers I–IV, we have more general directions of future work, not directly related to the articles.

In the current implementation of the model, we use a stochastic linear programming solver and a piecewise linear objective function. The reason behind this choice was that we did not believe that there existed solvers for convex, linearly constrained stochastic programming efficient enough to handle our problems. However, given progress in this area (see Steinbach [42] and Blomwall and Lindberg [8]), it would be interesting to apply a recursive interior point method to our problem.

In the current model, the company is allowed to trade only in asset classes. This has the disadvantage of requiring a large number of time-stages, as the risk–yield tradeoff may only be altered by trade. Mulvey, Madsen and Morin [36] have incorporated hybrid assets, such as investing in a trading strategy replicating a contingent claim (for instance a index-linked security with a guarantee). We would like to investigate this is a multi-stage stochastic programming setting, since this allows changing the investment portfolio between time-stages. This might help us mitigate the adverse effect of aggregating time-stages.

However, our main idea regarding future work regards parameterized policies. When creating an ALM model, we have to differentiate between the strategic and tactic levels of the model. At the strategic (that is, long-term) level, we determine which fractions of the available funds should be allocated to asset-classes, such as bonds and stock. At the tactic (that is, short-term) level we determine which stocks and bonds to buy. In our partner company this is implemented by the board making decisions on allocations into asset-classes, and for each asset-class, an account-manager determines which assets to hold in order to beat his target (generally an index given by the board).

In our model, we have relatively many asset classes compared to other models. We may see this as us including a larger part of the problem at the strategic level, compared to other models. In this framework, Paper III may be seen as an attempt to move this line up towards the strategic level. We do this by having broader, but fewer, asset-classes. However, how the assets are chosen is given by dual information from the strategic system. We may liken this to giving the account managers larger classes to manage, but instead of providing only an index as a benchmark, additional credit is given for a portfolio which positively correlates to the reserves. Mulvey [36] discusses this kind of framework. In his work, the strategic level chooses how to distribute the available funds over both assets and hybrid assets. The prices of different kinds of risk are obtained using dual information from the strategic level.

We would like to take this idea one step further, and totally remove the asset classes from the strategic level. We would do this in the framework of paper IV, using policy functions. Right now the policy function specifies which assets to hold, making this function unnecessarily high-dimensional (hence hard to optimize). We would instead like to test a more complex policy function, having two layers. The top layer will take the policy parameters and the state of the company as input and give the price of risk for an appropriate risk-measure (for instance 10% C-VaR), as well as the reward for letting yield on the asset-mix correlate with the reserves. The lower level would find the asset mix by taking the price of risk and optimize the risk-adjusted yield on the assets. In this lower level optimization, we would consider transaction costs and legal restrictions on the company. Hence, the upper level would only deal with which level of long-term risk to take, and how to distribute this risk over time and the state of the company. We believe that this would improve the policy-functions used in paper IV, as it would lower the dimension of the policy function, as well as treat transaction costs in a better fashion than the current no-trade region.

References

[1] F. Andersson, H. Mausser, D. Rosen, and S. Uryasev, *Credit risk optimization with conditional value-at-risk criterion*, Mathematical Programming, 89 (2001), pp. 273–291.

[2] P. Barth and A. Bockmayr, *Modelling mixed-integer optimisation problems in constraint logic programming*, Tech. Rep. MPI-I-95-2-011, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, November 1995.

[3] J. Birge, M. Dempster, H. Gassman, E. Gunn, A. King, and S. Wallace, *A standard input format for multiperiod stochastic linear programs*, COAL Newsletter, 17 (1987), pp. 1–19.

[4] J. R. Birge, *Decomposition and partitioning methods for multistage stochastic linear programs*, Operations Research, 33 (1985), pp. 989–1007.

[5] J. R. Birge, J. Donohue, Christopher, D. F. Holmes, and O. G. Svintsiski, *A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs*, Mathematical Programming, 75 (1996), pp. 327–352.

[6] J. R. Birge and F. V. Louveaux, *A multicut algorithm for two-stage stochastic linear programs*, European Journal of Operational Research, 34 (1988), pp. 384–392.

[7] J. R. Birge and L. Qi, *Computing block-angular Karmarkar projections with applications to stochastic programming*, Mangement Science, 34 (1988), pp. 1472–1479.

[8] J. Blomvall and P. O. Lindberg, *A Riccati-based primal interior point solver for multistage stochastic programming*, European Journal of Operational Research, 143 (2002), pp. 452–461.

[9] S. P. Bradley and D. B. Crane, *A dynamic model for bond portfolio management*, Management Science, 19 (1972), pp. 139–151.

[10] M. J. Brennan and E. S. Schwartz, *Alternative investment strategies for the issuers of equity linked life insurance policies with an asset value guarantee*, The Journal of Business, 1 (1979), pp. 63–93.

[11] D. R. Cariño, D. H. Myers, and W. T. Ziemba, *Concepts, technical issues, and uses of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 450–462.

[12] D. R. Cariño and W. T. Ziemba, *Formulation of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 433–449.

[13] D. R. Cariño, W. T. Ziemba, T. Kent, D. H. Myers, C. Stacey, M. Sylvanus, A. L. Turner, and K. Watanabe, *The Russell–Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming*, Interfaces, 24 (1994), pp. 29–49.

[14] D. Chambers and A. Charnes, *Inter-temporal analysis and optimization of bank portfolios*, Management Science, 7 (1961), pp. 393–410.

[15] A. Charnes and W. W. Cooper, *Chance-constrained programming*, Management Science, 6 (1959), pp. 73–79.

[16] C. Condevaux-Lanloy and E. Fragniere, *SETSTOCH: A tool for multistage stochastic programming with recourse*, tech. rep., Department of Management Studies, University of Geneva, 1998.

[17] G. Consigli and M. Dempster, *Dynamic stochastic programming for asset–liability management*, Annals of Operations Research, 81 (1998), pp. 131–161.

[18] M. Dempster and R. Thompson, *Parallelization and aggregation of nested Benders decomposition*, Annals of Operations Research, 81 (1998), pp. 163–187.

[19] ——, *EVPI-based sampling solution procedures for multistage stochastic linear programmes on parallel MIMD architectures*, Annals of Operations Research, 90 (1999), pp. 161–184.

[20] C. Dert, *Asset liability management for pension funds; A multistage chance constrained programming approach*, PhD thesis, Erasmus University Rotterdam, 1995.

[21] H. I. Gassman, *MSLiP: A computer code for the multistage stochastic linear programming problem*, Mathematical Programming, 47 (1990), pp. 407–423.

[22] J. Gondzio and R. Kouwenberg, *High performance computing for asset liability management*, Operations Research, 49 (2001), pp. 879–891.

[23] N. H. Hakansson, *Optimal investment and consumption strategies under risk for a class of utility functions*, Econometrica, 38 (1970), pp. 587–607.

[24] N. H. Hakansson, *Multi period mean variance analysis: toward a general theory of portfolio choice*, The Journal of Finance, 26 (1971), pp. 857–884.

[25] P. Hilli, M. Koivu, T. Pennanen, and A. Ranne, *A stochastic programming model for asset liability management of a Finnish pension company*, Stochastic programming E-print series, (2003). http://dochost.rz.hu-berlin.de/speps/.

[26] K. Høyland, *Asset liability management for a life insurance company: A stochastic programming approach*, PhD thesis, Department of Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway, 1998.

[27] K. Høyland, E. Ranberg, S. W. Wallace, and W. S, *Kapitalforvaltning i et livselskap*, Praktisk Økonomi og ledelse, (1997), pp. 71–84. In Norwegian. Also published as a part of [26].

[28] K. Høyland and S. W. Wallace, *Analyzing legal regulations in the Norwegian life insurance business using a multistage asset-liability management model*, European Journal of Operational Research, 134 (2001), pp. 293–308.

[29] J.P. Morgan, *RiskMetrics*, fourth edition ed., 1996.

[30] P. Kall and J. Mayer, *SLP-IOR: An interactive model management system for stochastic linear programs*, Mathematical Programming, 75 (1996), pp. 221–240.

[31] H. Konno and H. Yamazaki, *Mean-absolute deviation portfolio optimization model and its applications to Tokyo stock market*, Management Science, 37 (1991), pp. 519–531.

[32] M. Kusy and W. Ziemba, *A bank asset and liability management model*, Operations Research, 34 (1986), pp. 356–376.

[33] L. Lopes, *A modeling language for stochastic programming.* Conference paper, presented at ISMP2003, Aug 18–22, Copenhagen, 2003.

[34] H. Markowitz, *Portfolio selection*, The Journal of Finance, 7 (1952), pp. 77–91.

[35] J. M. Mulvey, G. Gould, and C. Morgan, *An asset and liability management system for Towers Perrin–Tillinghast*, Interfaces, 30 (2000), pp. 96–114.

[36] J. M. Mulvey, C. Madsen, and F. Morin, *Linking strategic and tactical planning systems for asset and liability management*, Annals of Operations Research, 85 (1999), pp. 249–266.

[37] X. C. Poiré, *Model generation and sampling algorithms for dynamic stochastic programming*, PhD thesis, Department of Mathematics, University of Essex, Colchester, U.K., 1995.

[38] R. T. Rockafellar and R. J.-B. Wets, *Scenarios and policy aggregation in optimization under uncertainty*, Mathematics of Operations Research, 16 (1991), pp. 119–147.

[39] A. Ruszczyński, *A regualrised decomposition method for minimizing a sum of polyhedral functions*, Mathematical Programming, 35 (1986), pp. 309–333.

[40] P. A. Samuelson, *Lifetime portfolio selection by dynamic stochastic programming*, The review of economics and Statistics, 51 (1969), pp. 239–246.

[41] W. F. Sharpe and L. G. Tint, *Liabilities-a new approach*, the journal of portfolio management, (1990), pp. 5–10.

[42] M. C. Steinbach, *Recursive direct algorithms for multistage stochastic programs in financial engineering*, tech. rep., Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1998.

[43] S. Takriti, B. Krasenbrink, and L. S.-Y. Wu, *Incorporating fuel constraints and electricity spot prices into the stochastic unit commitment problem*, Operations Research, 48 (2000), pp. 268–280.

[44] P. Valente, G. Mitra, C. Poojari, and T. Kyriakis, *Software tools for stochastic programming: A stochastic programming integrated environment (SPInE)*, tech. rep., Department of Mathematical sciences, Brunel University, Uxbridge, U.K., 2001.

[45] R. M. Van Slyke and R. Wets, *L-shaped linear programs with applications to optimal control and stochastic programming*, SIAM Journal on Applied Mathematics, 17 (1969), pp. 638–663.

[46] J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behaviour*, Princeton University Press, third ed., 1953.

[47] R. J. Wittrock, *Dual nested decomposition of staircase linear programs*, Mathematical Programming Study, 24 (1985), pp. 65–86.

A Obtaining a near-optimal dual solution from nested benders decomposition

In Paper I and III of this thesis we determine search-directions for a gradient-descent method using dual information from a linear stochastic programming problem. Hence we need a near-optimal dual solution to a multi-stage linear stochastic programming problem.

Consider the stochastic programming problem to

$$\begin{aligned}
\text{minimize} \quad & c^{\mathrm{T}}x + \sum_{k=1}^{n} p_k d_k^{\mathrm{T}} y_k, && \text{(A.1)}\\
\text{subject to} \quad & Ax = b, && \text{(A.2)}\\
& T_k x + W_k y_k = h_k, \quad k = 1,\dots,n, && \text{(A.3)}\\
& x \geq 0, && \text{(A.4)}\\
& y_k \geq 0, \qquad\qquad k = 1,\dots,n. && \text{(A.5)}
\end{aligned}$$

This problem has n scenarios each occurring with probability $p_k, k = 1,\dots,n$. The decision which has to be made before the random information is revealed, is denoted by x, and the corrective action by y.

The dual formulation of this problem is to

$$\text{maximize} \quad \rho^{\mathrm{T}} b + \sum_{k=1}^{n} \gamma_k^{\mathrm{T}} h_k,$$

$$\text{subject to} \quad \rho^{\mathrm{T}} A + \sum_{k=1}^{n} \gamma_k^{\mathrm{T}} T_k \leq c^{\mathrm{T}}, \qquad \gamma_k^{\mathrm{T}} W_k \leq p_k d_k^{\mathrm{T}}. \tag{A.6}$$

The multi-cut version of the L-shaped algorithm works as follows (see Birge and Louveaux [6]):

Step 0 Set $q = 0, S = 0$ and $V_k = 0$ for $k = 1, \ldots, n$.

Step 1 Set $q := q + 1$ and solve the master problem to

$$\begin{aligned}
\text{minimize} \quad & c^{\mathrm{T}} x + \sum_{k=1}^{n} p_k \theta_k, && \text{(A.7a)}\\
\text{subject to} \quad & Ax = b, && \text{(A.7b)}\\
& (\pi_k^v)^{\mathrm{T}} T_k x + \theta_k \geq (\pi_k^v)^{\mathrm{T}} h_k, \quad k = 1, \ldots, n,\\
& \qquad v = 1, \ldots, V_k, && \text{(A.7c)}\\
& (\eta^s)^{\mathrm{T}} T_{k(s)} x \geq (\eta^s)^{\mathrm{T}} h_{k(s)}, \quad s = 1 \ldots S, && \text{(A.7d)}\\
& x \geq 0. && \text{(A.7e)}
\end{aligned}$$

obtaining an optimal solution x^q. Here, (A.7c) are the optimality cuts, and (A.7d) are the feasibility-cuts, with $k(s)$ being the subproblem which did not have a feasible solution, resulting in the generation of feasibility-cut s. Information on how these cuts are generated is given in Step 2 and 3.

If $V_k = 0$ then the corresponding value of θ_k is fixed to 0 during the solution. If $q > 1$ and $x^q = x^{q-1}$, then terminate with the optimal solution.

Step 2 For $k = 1, \ldots, n$ solve the subproblem to

$$\begin{aligned}
\text{minimize} \quad & Q_k(x^q) = d_k^{\mathrm{T}} y_k,\\
\text{subject to} \quad & W_k y_k = h_k - T_k x^q, && \text{(A.8)}\\
& y_k \geq 0.
\end{aligned}$$

If this problem has no feasible solution for some k, then go to Step 3, otherwise take the dual solution π and set $V_k := V_k + 1, \pi_k^{V_k} = \pi$. This yields a supporting hyper-plane to the graph of Q_k: $Q_k(x) \geq (\pi_k^{V_k})^{\mathrm{T}}(h_k - T_k x)$, for all x, with equality holding for x^q. When all subproblems are solved, with no subproblem being infeasible, go to Step 1.

Step 3 If the subproblem above is infeasible for some k, then solve

$$\begin{aligned}
\text{minimize} \quad & \sum w_i^+ + w_i^-,\\
\text{subject to} \quad & W_k y_k + I w^+ - I w^- = h_k - T_k x^q,\\
& y_k, w^+, w^- \geq 0.
\end{aligned}$$

This, in effect, minimizes the L_1 norm of the distance to the feasible set. The dual solution, η, yields a hyper-plane cutting off the vector x^q as it does not fulfill $\eta^{\mathrm{T}} T_k x \geq \eta^{\mathrm{T}} h_k$. (In fact, all values of x making the subproblem (A.8) feasible must fulfill this inequality.) Now, set $S := S + 1, s(S) = k, \eta^S = \eta$ and go to Step 1.

At the terminal iteration of the nested Benders algorithm, the master problem has the form of equation (A.7). This problem has the corresponding dual problem to

$$\begin{aligned}
\text{maximize} \quad & \bar{\rho}^{\mathrm{T}} b + \sum_{k=1}^{n} \sum_{v=1}^{V_k} (\lambda_v^k (\pi_k^v)^{\mathrm{T}} h_k) + \sum_{s=1}^{S} \mu_s^{\mathrm{T}} (\eta_k^s)^{\mathrm{T}} h_{k(s)}, \\
\text{subject to} \quad & \bar{\rho}^{\mathrm{T}} A + \sum_{k=1}^{n} \sum_{v=1}^{V_k} (\lambda_v^k \pi_k^v T_k) + \sum_{s=1}^{S} (\mu_s)^{\mathrm{T}} T_{k(s)} \leq c, \\
& \sum_{v=1}^{V_k} \lambda_v^k = p_k, \quad k = 1, \dots, n, \\
& \lambda \geq 0.
\end{aligned} \tag{A.9}$$

Now, for some x, π_k^v is an optimal dual solution to

$$\begin{aligned}
\text{minimize} \quad & d_k^{\mathrm{T}} y_k, \\
\text{subject to} \quad & W_k y_k = h_k - T_k x, \\
& y \geq 0, \quad k = 1, \dots, n.
\end{aligned} \tag{A.10}$$

For some (other) x, η_k^s is an optimal dual solution to

$$\begin{aligned}
\text{minimize} \quad & \sum v^+ + v^-, \\
\text{subject to} \quad & W_{k(s)} y_k + I v^+ - I v^- = h_{k(s)} - T_{k(s)} x, \\
& y, v^+, v^- \geq 0.
\end{aligned} \tag{A.11}$$

We put

$$\gamma_k := \sum_{v=1}^{V_k} \lambda_v^k (\pi_k^v) + \sum_{s:k(s)=k} \mu_{k(s)} (\eta^s). \tag{A.12}$$

As we know that π_k^v and μ^s fulfill

$$\begin{aligned}
\pi_k^{\mathrm{T}} W_k &\leq d_k^{\mathrm{T}}, \\
(\eta^s)^{\mathrm{T}} W_{k(s)} &\leq 0,
\end{aligned}$$

since they are feasible dual solutions to (A.10) and (A.11), we get that

$$\gamma_k^{\mathrm{T}} W_k = \left(\sum_{v=1}^{V_k} \lambda_v^k (\pi_k^v)^{\mathrm{T}} + \sum_{s:k(s)=k} \mu_k (\eta^s)^{\mathrm{T}} \right) W_k \leq p_k d_k^{\mathrm{T}},$$

as equation (A.9) gives that

$$\sum_{v=1}^{V_k} \lambda_v^k = p_k.$$

By substitution, we have that $(\bar{\rho}, \lambda, \eta)$ fulfills the constraints of (A.9), and hence $(\bar{\rho}, \gamma)$ fulfills (A.6). As we have now found a feasible dual solution with an objective value equal to the primal objective value, we know that this dual solution is optimal. This result is useful from two perspectives. Firstly, the dual variables of the first stage are a part of an optimal dual solution to the whole problem, and secondly, in the absence of feasibility cuts, and with only one optimality cut active for a certain subproblem, the dual variables of this subproblem are also a part of a dual solution to the whole problem. This is true since in this case we have that

$$\gamma_k := \lambda_v^k \pi_k^v$$

in equation (A.12), with k being the active cut. As any multi-stage problem may be viewed as a two-stage problem, where all stages except the first one are considered to be the second stage, we see that the property of being a partial valid solution extends recursively to all single-cut children of only single-cut nodes.

If the number of active cuts is low, we hence get a dual solution, of which a large subset is a part of a valid dual solution.

These conditions may seem restrictive, but for the highly constrained problems solved in Paper I, almost always only one of the subproblems of a given node will have more than one active cut. Furthermore, as the dual variables of the master problem are always a part of a correct dual solution, aggregating stages to make the master problem larger will increase the subset of the total dual solution which is correct.

The reason that we may not use equation (A.12) to obtain the exact dual solution is that the implementation does not store the dual values π_k^v and η_k^s. If these values were saved for each cut, obtaining the full dual would be possible. If we do not have the space available to store the dual solution corresponding to each cut, we may still obtain a full dual. We do this by solving the problem to optimality, after which the constraints (A.7c) are relaxed for all k that have more than one active cut, and the constraints (A.7d) are relaxed for all active cuts. The cuts are relaxed by reducing the right-hand sides enough to trigger a regeneration of the cut. After this, the problem is resolved while retaining the dual information.

Paper I

An asset liability management system for a Swedish life insurance company

Fredrik Altenstedt
Department of Mathemathics
Chalmers University of Technology
412 96 Göteborg, Sweden

September 22, 2003

Abstract

When managing assets used for life insurance, an investor must make a trade-off between the long-term goal of high return over the customers' lifetime, and the short-term goal of fulfilling government regulations. In order to facilitate making this trade-off we have developed an asset liability management model for the Swedish set of laws and regulations. The most significant difference between our model and other models from the literature is that in the Swedish case a company's actions will not only influence the asset allocation, but also the reserve requirements. As the problem is dynamic in nature, we employ multi-stage stochastic programming to optimize the asset allocation. In order to determine the usefulness of the system developed, we perform tests using rolling horizon simulations, where the stochastic programming approach is compared to a fix-mix benchmark. We further test different shapes of the scenario tree. The results from these tests suggest that the practice of aggregating stages in an ALM model in order to tackle problems with many stages will give the solution a significant bias.

Keywords: Stochastic programming; asset liability management; Optimization; Finance; Life insurance

1 Introduction

The basic retirement benefit system in Sweden is tax-funded. Each employer pays a fraction of the salary of their employees to the government retirement funds, which provide benefits for the retired work force. Private insurance offered by life insurance companies exists as a complement to the public system. On the Swedish market there exist different kinds of private pension insurances, but the only one treated in this paper is the traditional type, where the company carries the main part of the investment risk. Under such a traditional plan, the customer pays an amount each month, and upon retirement he/she receives a monthly sum depending on the total amount paid and the success of the company's investments. However, there is a lowest level of return that the company must pay to the customer; hence a traditional pension insurance policy may be viewed as a defined contribution plan with an element of a targeted money purchase plan.

In most applications of financial asset management the investor tries to choose investments giving good returns while minimizing the risk, where risk is measured as the variance or downside moments of investment returns. For an insurance company, such risk measures ignore the purpose of the investments, which is to be able to honor the company's liabilities. Hence, a more appropriate risk measure for an insurance company is the expected deficit, measuring the risk that the company does not own enough assets to cover the customers' claims when they occur. *Asset liability management* (ALM) deals with the unified administration of assets and liabilities, and may provide a strategy more suitable for a life insurance company than the separate management of assets and liabilities.

In this paper we develop an asset liability management system for a Swedish life insurance company. The system consists of a mathematical model of the company's assets and liabilities, a mathematical model of the surrounding economy and a stochastic programming solver capable of solving large instances of the model. In Section 2 we give an overview of stochastic models for ALM, with a focus on models used for retirement benefits. In Section 3 we provide an overview of the model of the company and the surrounding economy. Finally, we present tests of the model in Section 4, and conclusions are drawn from these tests in Section 5.

2 Overview of stochastic programming models for asset liability management

Ever since Dantzig [11] first considered extensions of ordinary mathematical programming problems to accommodate stochasticity in the data, a number of researchers have used stochastic programming to address problems for which decisions have to be made from incomplete information. In most of these models, the underlying stochasticity has a continuous distribution. As continuous distributions are hard to handle computationally, the distributions are discretized in order to obtain a finite number of possible outcomes. If this discretization is to accurately mimic the underlying continuous distribution, a large number of discretization points are needed. Hence the resulting problems tend to be large and require significant computational resources to find optimal solutions. This is especially true for multi-period problems, for which decisions and observations are interspersed; the size of such problems grows exponentially with the number of stages. This computational burden made applications of stochastic programming few and far in between until computing power became more ubiquitous and affordable.

As risk and the stochasticity underlying risk is most central to the financial industry a large number of stochastic programming models have been developed for financial applications. Bradley and Crane [2] developed a model for managing bond portfolios as well as a decomposition solver for the (at the time) large resulting stochastic linear programs. Kallberg, White, and Ziemba [21] created a two-stage model to manage short-term cash flows; the model was later extended to several time periods by Kusy and Ziemba [23]. Other uses of stochastic programming for financial problems include [6, 5, 7], where the authors use stochastic programming to improve the asset liability management of a Japanese insurance company, as well as a number of models dealing with ALM for pension funds and life insurance companies, described further below.

2.1 Swedish life insurance rules

The problem faced by a life insurance company is to optimally manage its assets while at all times complying with the current laws and regulations. Hence, a short description of these laws is needed in order to formulate the company's problem mathematically.

2.1.1 Rates of return

Under Swedish law there is a lowest rate of return that the holder of a life insurance policy must receive on his/her savings. This guaranteed rate of return is set by the regulating authorities; its level is however influenced by the long-term market interest rates. In addition to this lowest rate of return, excess funds are distributed among the customers using a bonus rate of return, which is set by the company. Hence, each contract may be viewed as an ordinary bank account accruing interest given by the bonus rate. This means that the company itself influences its future reserve requirements by setting their bonus rate. Since there is an element of guaranteed return, a Swedish life insurance policy may be viewed as a target money purchase plan. The policy is however not entirely consistent with such a plan, as the guaranteed rate of return may change during the plan's lifetime.

2.1.2 Reserves

In order to guarantee the solvency of a life insurance company, laws require that a life insurance company owns assets in excess of two reserve levels. The *retrospective reserve* is computed as the sum of all payments made to the company, increased with the bonus rate of return accrued so far. If a customer of age x makes a one-time deposit of D_0 at time 0, the retrospective reserve V for this deposit at time t will be given by

$$V(t) = D_0(1-\rho)\text{exp}\left(\int_0^t \delta + (\alpha\mu(x+\tau) - \beta)\, d\tau\right), \tag{2.1}$$

where δ is the bonus rate intensity, $\mu(x)$ is the death intensity of a customer (the probability that a customer will die in a short interval t to $t + \Delta t$ is given by $\mu(x)\Delta t$), ρ is a deposit fee and α and β are security factors. When the reserves are calculated, an insurance policy is viewed as a collection of one-time payments, and the reserve requirement for the company is found by summing the reserve contributions from all payments to the company by their customers.

The *prospective reserve* is given by the present value of the lowest allowed future payments given by the guaranteed rate of return. In order to illustrate this reserve, we assume that we have a customer of age x who has a policy giving a guaranteed continuous stream of 1 SEK per year. The stream will be paid out m years from now and and continue for r years, given that the customer is still alive. The present value of this stream will be

$$K(x, m, r) = \int_{m}^{m-r} e^{-\bar{\delta}} \tau P(T_x > \tau)\, d\tau,$$

where $\bar{\delta}$ is the interest rate intensity of the guaranteed rate of return offered to the customer, and $P(T_x > \tau)$ is the probability that the remaining lifespan of a person of age x is larger that τ. If our customer now makes a one-time deposit of D_0, this will buy her a guaranteed continuous stream of g SEK per year, with g being defined by

$$g := \frac{D_0}{(1+\rho)K(x, m, r)},$$

although once she reaches her retirement age, the actual payment stream received may be higher, as given by the bonus rate above. Here, again, ρ is a deposit fee. The prospective reserve for this one time deposit will be

$$S = gK(x, M, r),$$

but now the present value K has to be computed using the lowest allowed guaranteed rate of return. The lowest allowed guaranteed rate of return is set by the regulating authority *Finansinspektionen* (Swedish for the Finance Inspection), and it is affected by the long-term market rates. The promise made to the customer, g, is fixed and independent of the rate used to compute it, which means that if the guaranteed rate of return is changed, the present value of a future payment stream, $K(x, m, r)$ will change, and so will the prospective reserve. Hence there is a coupling between the asset prices and the liabilities via the long-term market rates.

2.1.3 Reserve requirements

The consolidation of the company is defined as the sum of the value of all assets divided by the retrospective reserve (defined above). The regulating authorities require that a life insurance company keeps its consolidation between 100% and 120%. These limits are not strictly enforced: consolidation levels below 100% are accepted for shorter periods of time. However, if this requirement is violated during long periods of time, or by a significant amount, authorities may require the company to retroactively lower the bonus rate of return. Retroactively reducing the bonus is perfectly legal, but very undesirable from a public relations point of view.

In addition to covering the retrospective reserve, the company is required to own assets in excess of the prospective reserve. Furthermore, just covering this reserve is not enough; there are limits on which type of assets may be used to cover the reserve. For example, a company may cover at most 25% of the reserve using equity. (This does not limit the amount that a company may invest in equity, as long as the company owns at least 75% of the value of the prospective reserve in other assets.) As the prospective reserve deals with guaranteed benefits, the requirements related to this reserve are enforced in a more stringent way than the requirements related to the retrospective reserve: breaking these rules may result in the liquidation of the company.

2.2 Comparison to models under foreign rules

In order to highlight the differences and similarities between the model presented in this paper and other models used for ALM, a short overview of rules and regulations of other countries is given next.

2.2.1 Dutch conditions

The Netherlands has a pension system based largely on funded assets. As a consequence, large assets are held by Dutch pension funds (an estimated 25% of all the pension assets in the EU, according to Boender [1]) and the massive funds held by Dutch funds have inspired much research into ALM solutions under the Dutch set of rules.

Dert [13, 14] has developed an asset liability model for a Dutch pension fund, with chance constraints regulating the probability of under-funding. The problem addressed is that of managing assets for a fund set up by a group of companies (the sponsors of the fund) in order to provide retirement benefits for the sponsoring companies' employees. The benefits are dependent on the employees' final and average salary, and the job of the manager of the fund is to keep the fund sufficiently solvent while keeping the contributions low and predictable. In order to do so the objective is to minimize the expected present value of the contributions required to keep the fund sufficiently solvent, with an added penalty for remedial contributions. The resulting model is a mixed integer linear problem, which is solved via a heuristic method. Dert models the economy via a VAR (Vector Auto Regressive) model, and the status of the beneficiaries of the fund is modeled by Markov chains.

In [17], Kouwenberg and Gondzio use the same framework, but replace the probability constraints by penalties for under-funding, and impose restrictions on the yearly variations of payments made to the fund. Again, the objective is to minimize the present value of the contributions, while penalizing remedial contributions.

The same framework is used once again by Kouwenberg [22], where different scenario generation models are examined. The model is identical to the one described by Gondzio and Kouwenberg [17] with the exception of applying a quadratic penalty on the remedial contributions.

The most notable difference compared to the Swedish case is the fact that the Dutch funds administer a defined benefits scheme (that is, participants receive a pension based on their final salary, and the contributions vary in order to achieve this), while the problem we address is mainly a defined contribution scheme (the contributions are defined by the customer, and the customer gets a pension dependent on the fund's development).

2.2.2 The Norwegian problem

Høyland and Wallace [20, 19] have treated the asset liability problem of a life insurance company in Norway. It is similar to the Swedish case: individuals save up for their own retirements, as opposed to the Dutch case, where the funding is provided by the employer. The case considered is hence a defined contributions scheme. The major difference from the Swedish setting is that all reserve requirements are considered to be exogenous variables (that is, they are not affected by the company's actions). Norway has a requirement of a lowest guaranteed rate of return on the assets, similar to the Swedish case. These rules do however differ from the Swedish ones as the guaranteed rate of return has to be given annually in Norway, as opposed to over the lifetime of the contract in Sweden. Requiring annual yields is a very limiting constraint, severely reducing the possible investments. Furthermore, Norwegian life insurance customers has the right to move their contract to a competing firm by paying a nominal fee (i.e. the contract contains a surrender option), a right the Swedish customers lack. Taking these rules into account, Høyland models the problem by constructing a multi-stage stochastic linear programming model, with a piecewise linear convex objective function.

2.2.3 Other nationalities

Asset liability management problems for pension management have been addressed by a number of other researchers. Pflug et al. [25, 27, 26] consider the problem in an Austrian setting. The model is a multi-stage linearly constrained problem, with different objective functions in different versions. It is solved using interior point methods, taking advantage of the special structure of the constraint matrix for efficient factorization.

A similar problem for a British fund is addressed by Dempster and Consigli [8], who formulated the general purpose CALM model. This model differs from the other models described mainly by making it possible to add capital gains taxes, which requires the model to keep track of not only how much of each asset is owned, but also when it was purchased. Another difference is the absence of reserve requirements; liabilities need only be covered when they occur. The model is a multi-stage stochastic linear programming model, although versions with quadratic penalties exist.

An alternative to using multi-stage stochastic programming is explored by Brennan and Schwartz [3], who treat investment for index-linked policies with guarantees, a situation slightly similar to ours. In their framework, the payments made to the customers may be modeled as a combination of a guaranteed amount and a call-option on the index, with the guaranteed amount as the strike-price. under this framework, investment strategies are found by replicating the option. However, this framework is not very appropriate for our problem, as the company in our case influences the amount payed to a large degree, by setting the bonus rate of return.

3 The ALM model

In this section we describe the mathematical model of the company and the surrounding economy. In order not to explicitly denote all dependencies of random information, parameters and variables which vary between scenarios will be indicated with a boldface font.

3.1 Modeling of exogenous variables

We have modeled seven asset classes for the company to invest in, chosen based on our partner company's current asset classification. These assets are Swedish and foreign bonds, Swedish and foreign stock, Swedish treasury bills, Swedish real interest rate bonds, and Swedish real estate.

Interest rates and bond prices In our model of a Swedish life insurance company, the interest rates influence both the yield on bonds and the reserve requirements (see Section 2.1.1). This means that the interest rates have a high impact on which decision is optimal. Hence, bond prices and interest rates must be modeled in a consistent way. This is done by employing a version of the Brennan–Schwarts interest rate model [4]. This model has two state variables: the instantaneous interest rate and the rate of return on a console bond (a bond which never matures). These rates of returns are simulated by time-stepping a stochastic differential equation, and the price of any zero-coupon bond may be obtained by numerically solving the bond pricing equation (see Appendix B for details). This procedure is used to obtain prices of treasury-bills and bonds consistent with the interest rates.

A two-factor model is chosen since preliminary studies indicated that the one-factor models tested (the Vasicek model [9] and the Cox-Ingersoll-Ross model [10]) yielded too high correlations between long and short interest rates. As console bonds are not traded on the Swedish market, we have used the interest rate of the longest government bond on the Swedish market when fitting the model to market data. The parameter values obtained are given in Table 6 in Appendix A.

Other asset prices Inspired by the capital asset pricing method, we assume that the expected yield on assets other than Swedish bonds and treasury-bills are given by the risk-free rate of return to which a risk premium is added. The only exception is long-term foreign bonds where the expected yield is taken as the same as the expected yield on Swedish bonds, in effect assuming that the real long-term interest rate in and outside of Sweden are comparable. We generate the returns for these assets conditionally on the returns given on Swedish bonds and Swedish treasury bills, in order to get the desired correlation between the different asset classes. The parameters used for scenario generation in this study are given in Tables 7 and 8 in Appendix B.

The parameter values used in the simulation are a mixture of historical data and reasonable guesses, as the parameters used should not reflect the past, but the company managers' expectations about the future. Hence, historical data is used only to estimate correlations. In this article we only compare differences in performance between different configurations of the model itself. If the model is to be used commercially, or if a comparison is to be made versus our partner company's historical behavior, a better estimation of these parameters is needed. Especially, the values concerning Swedish real interest bonds and Swedish real estate are given by estimates from people within our partner company, and not based on real-world data. This is currently not a serious problem, since company policies prohibit trade in these assets, and hence trade in these assets is prohibited also in our model. The reason why these assets are not entirely excluded from the model is that these assets may still be used to cover the reserves, hence influencing which other asset allocations we may be interested in making.

Customers and liabilities As mentioned before in Section 2.1.3, the company needs to consider two reserves, one influenced by the bonus rate of return, and one entirely exogenous. A complication is that the retrospective reserve is not a linear function of the bonus rate, as may be seen from equation (2.1).

In order to be able to retain a linear ALM model, the calculation of the retrospective reserve needs to be slightly simplified. We do this by linearizing the retrospective reserve around an assumed value of the bonus rate. We believe that linearizing the reserve computations in this way will not greatly impact the solution, as the nonlinearity of the reserve is rather weak. As an example, we use one value of the bonus rate for the first 6 months, and another value for the next 18 months. In this case, the maximum difference between the linear approximation and the true reserve value will be less than 0.3% when the bonus rates are kept between 3% and 15%. (In previous trials, we have determined the reserves in an iterative fashion, by solving the ALM-problem in order to obtain the bonus rate of return for each node in the scenario tree, which have been used to update the expansion, after which the problem was resolved. The solutions obtained in this fashion did not significantly deviate from the solutions obtained without iterating.)

The prospective reserve is only influenced by the regulating authorities, and may be computed in a straightforward way. It is still dependent on the state of the economy, as the size of this reserve is determined by the guaranteed rate of return, which is influenced by the long-term interest rates.

For this study, we have used real-world data of the current stock of customers, augmented with our partner company's assumptions on the future inflow of customers. The customers have been aggregated according to year of birth, and the number of customers is assumed to be so large that the fraction of customers dying each year is exactly given by the mortality model.

A further assumption needed to make the model linear is that the customers do not react to the actions of the company, although the savings level of the customers may be related to other exogenous economic variables. It would be possible to treat the effects of customer behavior by linearizing how the consumer collective reacts to a change in the bonus rate of return, but as we have no data on the customers' reaction to a change in bonus rate, no such effort has been made.

Implementation of the economy model One of the goals when implementing the model of the economy was to make it easy to change or replace it if it was found to be insufficient, for instance by replacing the Brownian motions driving the model by a database of historical samples. In order to make this possible, the stochastic model is implemented in a black-box fashion as a pair of subroutines. The first of these routines take the two interest rates and a time length as input and return a pair of antithetic sample-paths of the desired time length, where the starting state is given by the two interest rates. The second routine returns the first four moments and the covariances of how the random variables develop over the given time length. If these moments are not explicitly available, they are found by simulation, by using a large number of samples from the first routine.

Antithetic samples are used as we wish to lower the errors in the sample means. As described earlier, the scenarios are generated using a stochastic differential equation, driven by an uncorrelated multivariate Brownian motion X. If we have a sample-path x of this Brownian motion, used to generate a sample-path $y(x)$ of the economic variables, an equally probable outcome of X is $-x$. We may use this sample to generate another sample-path $y(-x)$ from the distribution of economic variables. As $y(x)$ and $y(-x)$ are negatively correlated the sample mean gets a lower variance. The effects of using sample means for scenario tree generation have been investigated by Higle [18], who report that the method provided significantly better solution stability than ordinary random sampling.

3.2 Constraints

The model of the company is implemented as a multi-stage stochastic linear programming problem, having the following constraints:

Time linking constraints The development of the assets held are given by the time linking constraints, the assets owned in the first stage are given by what is held initially, corrected according to the first buy and sell decision:

$$\boldsymbol{x}_i^0 = \boldsymbol{y}_{+i}^0 - \boldsymbol{y}_{-i}^0 + \bar{x}_i, \quad i \in I. \tag{3.1}$$

Here, the different asset classes into which we may invest are given by I, and the assets held at time 0, $\boldsymbol{x}_i^0$, are given as the initial assets $\bar{x}_i$ to which we add what is bought, $\boldsymbol{y}_{+i}^0$, and deduct what is sold, $\boldsymbol{y}_{-i}^0$. In the same fashion, the asset inventory at later stages are given by

$$\boldsymbol{x}_i^t = \boldsymbol{y}_{+i}^t - \boldsymbol{y}_{-i}^t + (1 + \eta_i^t)\boldsymbol{x}_i^t, \quad i \in I, t \in T \setminus \{0\}, \tag{3.2}$$

where the price development of asset class i since the last trading time is given by η_i^t.

In order to simplify the constraints regarding the reserves, we also introduce a variable giving the total value of what the company owns

$$\boldsymbol{x}_{tot}^t = \sum_{i \in I} \boldsymbol{x}_i^t, \quad t \in T. \tag{3.3}$$

Cash balance Naturally, the sum of all transactions in a period must add up to 0, which is guaranteed by the cash balance constraint:

$$\begin{aligned} &\boldsymbol{P}_{\text{in}} - \boldsymbol{P}_{\text{out}} - \boldsymbol{\theta}^t \boldsymbol{x}_{tot}^t \\ &+ \sum_{i \in I} \left(\boldsymbol{y}_{-i}^t (1 - \gamma_i) - \boldsymbol{y}_{+i}^t (1 + \gamma_i) \right) + \sum_{i \in I} \boldsymbol{\rho}_i^t \boldsymbol{x}_i^{t-1} = 0, \quad t \in T. \end{aligned} \tag{3.4}$$

Here, the payments to the customers, $\boldsymbol{P}_{\text{out}}$, the payments from customers, $\boldsymbol{P}_{\text{in}}$, the tax payments and the net of the transactions and direct income (given by $\boldsymbol{\rho}_i^t$) must add up to 0. The taxes paid by a life insurance company in Sweden is proportional to the assets owned and a benchmark interest rate, the state borrowing rate (a weighted average of long-term government bonds). According to Swedish rules, each year the company pays 15% of what they would have earned as capital gains, had their assets been invested at the state borrowing rate. This is modeled by the factor $\boldsymbol{\theta}^t$, which is defined as the length of the period leading up to time t times 0.15 times the average state borrowing rate during this period. Furthermore, transaction costs for asset i are given by γ_i, and transaction costs are assumed to be proportional to the amount bought or sold. Furthermore, the direct yield of asset i during the period leading up to time t is given by ρ_i^t.

Bonus rate The bonus rate given to the customers is the primary way in which the different life insurance companies compete. Hence the company whish to hold this rate high, and preferably even over time. Right now we do not enforce the requirement that the rate should be even, but only try to avoid low rates of bonus return. As having a life insurance policy should be an attractive alternative compared to using a high interest account, we define offsets from the long-term interest rate as $\Delta r_{\text{ref},a}, a \in A$, and add penalties for having bonus rates below these levels:

$$\boldsymbol{r}^t + \boldsymbol{z}_a^t \geq \boldsymbol{r}_{ref}^t + \Delta r_{\text{ref},a}, \quad t \in T, a \in A. \tag{3.5}$$

Prospective reserve requirements The rules for a life insurance company specify that it must at all times be able to cover the prospective reserve using the correct types of assets, as described in Section 2.1.3. There is a limit to how much of the reserve may be covered using assets of a specified type. For instance, a maximum of 25% of the reserve may be covered using stock. In order to capture these requirements, we introduce variables $\hat{\boldsymbol{x}}_i^t$ stating how much of each asset class is used to cover the prospective reserve, (naturally we have $\hat{\boldsymbol{x}}_i^t \leq \boldsymbol{x}_i^t$). Right now only constraints on two sets of asset classes are relevant for the model, but we formulate these requirements more generally should the company decide to expand the model later on (for instance by including corporate bonds). Here K represents the set of rules. For each rule k there is a set I_k of assets affected by this rule, and a maximum fraction of the reserve, c_k, which may be covered by these assets. We hence get the upper bound on assets used to cover the reserve as

$$\sum_{i \in I_k} \hat{\boldsymbol{x}}_i^t \leq c_k \boldsymbol{S}^t, \quad t \in T, k \in K^t, \tag{3.6}$$

where $\boldsymbol{S}^t$ is the value of the prospective reserve at time t.

Failure to cover the reserve using the correct assets is a grave violation of the regulations, and will lead to the liquidation of the company. As liquidation is an alternative, although not a pleasant one, we do not strictly enforce this rule. We do instead add a steep penalty for failing to cover the reserve using the correct assets, a penalty given proportionally to the violation $\boldsymbol{z}_p^t$:

$$\sum_{i \in I} \hat{\boldsymbol{x}}_i^t + \boldsymbol{z}_p^t \geq \boldsymbol{S}^t, \quad t \in T. \tag{3.7}$$

As failing to cover the prospective reserve is a grave violation of the rules, almost failing to cover the prospective reserve is undesirable. We hence add a set of increasing penalties for almost failing to cover the reserve. In order to enforce avoiding violating this reserve requirements, we introduce a set of security levels $f_q, q \in Q$, where f_q ranges from 1.15 down to 0.9. We add levels below 1 even though the company goes bankrupt if the condition is violated, as the company may ask for extra contributions from the owners in order to avoid bankruptcy, and these contributions should be held small. The degree to which the security levels are violated are given by $\boldsymbol{z}_q^t$, defined by

$$\boldsymbol{x}_{tot}^t + \boldsymbol{z}_q^t \geq f_q \boldsymbol{S}^t, \quad t \in T, q \in Q. \tag{3.8}$$

Note that this expression uses the total assets $\boldsymbol{x}_{tot}^t$, not the sum of $\hat{\boldsymbol{x}}_i^t$. If the assets $\hat{\boldsymbol{x}}_i^t$ were used in the expression (3.8) this would force us to cover 115% of the reserve *using the correct asset classes*, something which is not required by law.

Retrospective reserve requirements The retrospective reserve is defined in order to guarantee that the company is on track to live up to the bonus rate promises given to the customers, as described in Section 2.1.3. We model failing to cover the retrospective reserve in the same fashion as the prospective reserve, using progressively steeper penalties. Hence we define a set of lower limits, $\kappa_{\min,m}$, $m \in M$ for a set M of penalties, and define the violation as previously by:

$$\boldsymbol{x}_{tot}^t + \boldsymbol{z}_m \geq \kappa_{\min,m} \boldsymbol{V}^t, \quad t \in T, m \in M. \tag{3.9}$$

In addition to preventing a too low consolidation, the regulating authorities wish a company to keep the consolidation under 120%, although this is not strictly enforced. (The idea is that gains made should be distributed to the policy-holders, and not retained in the company.) Hence we add a small penalty for having a too high consolidation as

$$\boldsymbol{x}^t_{tot} - \boldsymbol{z}_k \leq \kappa_{\max} \boldsymbol{V}^t, \quad t \in T. \tag{3.10}$$

Trading restrictions Our partner company has a policy of not actively trading real interest bonds and real estate. As for other assets, the company is small enough not to affect the market, and hence no trading restrictions are needed for other assets. However, for model generality, we still add upper bounds u_i for the trade of all assets, and set maximum trade high for all assets but real estate and real interest bonds.

$$\boldsymbol{y}^t_{+i} \leq u_i, \quad i \in I, t \in T, \tag{3.11}$$

$$\boldsymbol{y}^t_{-i} \leq u_i, \quad i \in I, t \in T. \tag{3.12}$$

As the rules for life insurance companies does not allow short selling, we add lower limits for all assets.

$$\boldsymbol{x}^t_i \geq 0, \quad i \in I, t \in T. \tag{3.13}$$

Expansions The value of the retrospective reserve, and the sum paid to the customers, are affected by past values of the bonus rate of return. As mentioned previously, we linearize the reserve and the payments, giving us the following expressions:

$$\boldsymbol{V}^t = \bar{\boldsymbol{V}}^t + \sum_{\tau=0}^{t-1} \frac{\partial \bar{\boldsymbol{V}}^t}{\partial \bar{\boldsymbol{r}}^\tau} (\boldsymbol{r}^\tau - \bar{\boldsymbol{r}}^\tau), \quad t \in T, \tag{3.14}$$

$$\boldsymbol{P}^t_{\text{out}} = \bar{\boldsymbol{P}}^t_{\text{out}} + \sum_{\tau=0}^{t-1} \frac{\partial \bar{\boldsymbol{P}}^t_{\text{out}}}{\partial \bar{\boldsymbol{r}}^\tau} (\boldsymbol{r}^\tau - \bar{\boldsymbol{r}}^\tau), \quad t \in T \tag{3.15}$$

In this expression, $\bar{\boldsymbol{V}}^t$ is the value the retrospective reserve would have if the bonus rates at previous periods had been $\bar{\boldsymbol{r}}^\tau, \tau < t$. In the same fashion, $\bar{\boldsymbol{P}}^t_{\text{out}}$ is the payments to the customers computed using the assumed values of the bonus rate of return.

3.3 Objective of the optimization

According to a recently abolished law, all Swedish life insurance companies had to be mutual companies, dividing all their profits among the customers. As our partner company does not plan to change their status from being a mutual company, we may assume that this restriction still applies. Since the company is mutual, a reasonable view is that the customers own all the money inside the company. Hence it is appropriate to maximize the expected value of the assets held by the company, as it is the customers' money. To the value of the company, benefit payments made to the customers must be added. In order not to skew the results and favor high inflation scenarios over low inflation scenarios, all results are discounted using the rate of inflation. In addition to maximizing the present value of the participants' money, the management of the company must take into account the rules and regulations imposed by the authorities. This is captured in the model by penalizing the deviations from certain goals, as described above.

As expressed above, there are rules that the assets owned must cover two reserve levels, with increasing penalties for increasing violations. Hence the objective will be a concave function of the total assets owned, making the company act in a risk-adverse fashion.

Thus we get the objective function, shown in equation (3.16), as the expected present value of all payments made to the customers, plus the expected terminal value of the company's assets reduced with penalties for violating different rules and regulations. The objective of the optimization is to maximize

$$\begin{aligned} w(x,z,r) \quad &:= \mathsf{E}[\boldsymbol{d}^{\mathrm{T}}\boldsymbol{x}_{tot}^{T} + \sum_{t\in T}\boldsymbol{d}^{t}\boldsymbol{P}_{\mathrm{out}}^{t} + \sum_{t\in T} l^{t}\boldsymbol{d}^{t}(-s_{p}\boldsymbol{z}_{p}^{t} - \sum_{a\in A} s_{a}\boldsymbol{z}_{a}^{t}\bar{\boldsymbol{V}}^{t} - \\ &\quad s_{k}\boldsymbol{z}_{k} - \sum_{m\in M} s_{m}\boldsymbol{z}_{m} - \sum_{q\in Q} s_{q}\boldsymbol{z}_{q})]. \end{aligned} \tag{3.16}$$

In this expression, the parameter l^t gives the length of the period following time t, which is used to scale the penalties. Penalties for the constraint violations defined in equations (3.5)–(3.10) above are given by s_p, s_a, s_k, s_m, s_q. Now the optimization problem may be stated as the maximization of (3.16) under the constraints (3.1)–(3.15).

4 Numerical tests

4.1 Rolling horizon simulations

Naturally, we wish to determine if our proposed model gives useful advice on how to run a life insurance company. As observing objective values does not give any information on how good a solution process is when used iteratively, we use rolling horizon simulations similar to Kouwenberg [22], Fleten, Høyland and Wallace [15], and Golub et al. [16], to test the performance of the method. The major difference between our tests and the ones performed in the cited articles is that they all treat tests performed where the number of stages in the test scenarios and the scenario trees are the same. (Kouwenberg uses a five period (six stage) tree whereas Fleten et al. use a three period tree). We use test scenarios with ten periods, forcing us to aggregate time-stages in the ALM model. This makes it possible for us to test if there is any performance difference between trees with few stages and many branches per stage, and trees with many stages but few branches at each stage.

Rolling horizon simulations work by applying the decision given by the optimization routine to our model of the company, letting some simulated time pass, an apply optimization again to obtain a new decision. The process is repeated and thus simulates how well the company actually fares when it is governed by the optimization model. These simulations are carried out as follows: First we generate a set of sample-paths which will be used for the evaluation, with all these sample-paths originating in a common state of the world. In order to run a scenario of length T, where the solution is evaluated each n time-steps, we apply the following procedure to each path.

0: Set $t = 0$ and go to step 2.

1: Use the sample-path and the state of the company just after the decision at time $t - n$ to generate the state of the company just before a decision is made at time t.

2: Use the state of the world of the current sample-path at time t to generate a scenario tree. Note that this tree is independent of the future realization of the sample-path, and hence that information does not leak into the scenario generation procedure.

3: Optimize over the tree, generating a decision for the company at time t. Use this decision to determine the state of the company after the decision is made at time t. Store information about the state of the company just after the decision.

4: if $t < T$, set $t = t + n$ and go to 1.

5: Use the stored states of the company to determine total penalties and the terminal value of the company, which are used to evaluate the success of a method on a particular scenario.

This procedure is repeated for each strategy tested, and the results are compared on a scenario to scenario basis. An illustration of the testing procedure is given in Figure 1.

The success of the method for a specific scenario is measured by taking the terminal value of the assets held by the company, to which we add all payments made to the customers, and subtract the penalties incurred. All values are discounted using the rate of inflation. Comparing the performance of two methods for one scenario will give us one sample of a random variable defined as the difference between using the two methods on a random scenario. As each scenario will give us one sample, we may use these samples to test whether the average value of the difference significantly differs from 0.

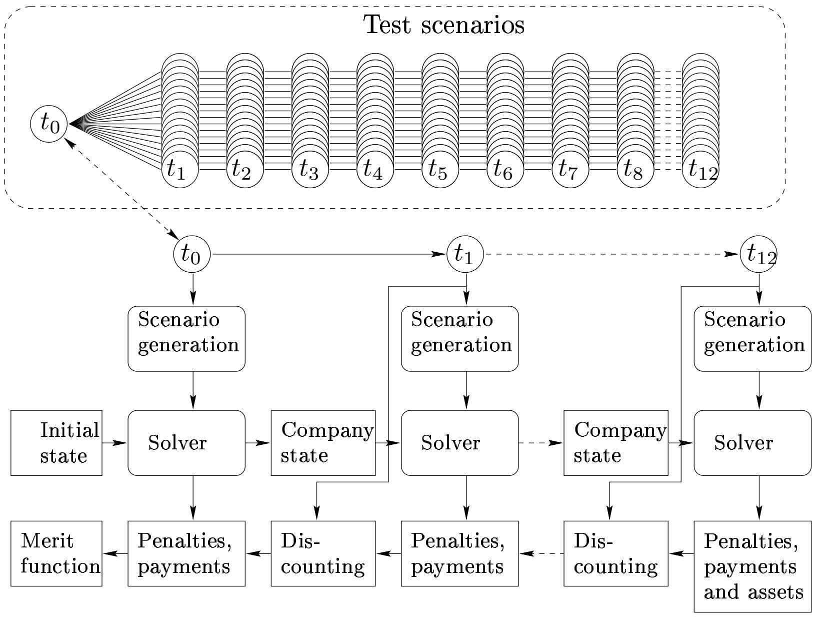

Figure 1: Testing system.

4.2 Scenario tree generation

In this work we employ two techniques for generating scenario trees: fitting and random sampling. When random sampling is used, the tree is generated recursively by sampling possible outcomes for the children of a node, using the model of the economy. In order to improve the sampling procedure, we use antithetic sampling and translate the samples obtained so that they will have the same expected mean as the sampled distribution.

When we do fitting the tree is again generated recursively, but by simultaneously optimizing the values of interest rates, asset prices and probabilities of the children of a specified node, in order to match the statistical properties of the underlying distribution. The variables fitted are the long and short interest rate, as well as asset prices for all assets except Swedish bonds and treasury-bills. Prices for Swedish long and short bonds are later given by the values of the interest rates. When using an optimized scenario tree generation, we fit the mean, variance, skewness and kurtosis of each variable, as well as the covariances between the different variables.

4.3 Optimization of fix-mix strategies

The current method used by our partner company to determine their operating strategy is hard to describe in mathematical terms. The board meets with even intervals to determine which asset mix the company should hold, and how high the bonus rate should be. The asset mix is given as a lower and upper bound on the fraction of the total wealth invested in different asset classes. When deciding on which asset mix to choose one naturally considers the consequences of keeping this asset mix for an extended period of time. Hence an approximation of this strategy is to choose the fixed mix of assets which will give the best yield over the period in question.

4.3.1 Fix-mix evaluation

In order to evaluate the effects of keeping one fixed asset mix, we add constraints to the model described in Section 3 and solve it. These constraints are

$$\alpha_i x^t_{\text{traded}} = x^t_i, \qquad i \in \bar{I}, \qquad i = 1, \ldots, T \tag{4.1}$$

where we use the following notation.

x^t_{traded}	total assets owned at time t, except assets invested in real estate and real interest bonds,
$\bar{I}$	Set of all assets except real interest bonds, real estate and Swedish bonds,
α_i	fix-mix fraction of asset i, $\alpha_i \geq 0, \sum_{i \in \bar{I}} \alpha_i \leq 1$.

No prescribed fraction is set for Swedish bonds in order to avoid linearly dependent constraints. With this setup, Swedish bonds will absorb the remainder of the funds invested, whenever $\sum_{i \in \bar{I}} \alpha_i < 1$.

This extended model is solved using the same procedure as the ordinary ALM model. Naturally, there are more efficient methods to evaluate a fixed mix. Our method however has the advantage of using the same method to determine the bonus rate of return in both models.

The fix-mix asset fractions are optimized using a gradient descent algorithm. Gradients with respect to asset fractions are obtained from the dual variables corresponding to the constrains (4.1), in the manner described in Dantzig and Thapa [12] on page 196. These gradients may only be obtained if x^t_{traded} is a basic variable. The total assets owned will most probably always be positive, and we indeed encountered no case for which $x^t_{\text{traded}} = 0$ occurred (which otherwise would have resulted in the evaluation being aborted).

Note that finding a fixed asset mix is not necessarily a convex problem (see Maranas et al. [24]), but in our case the method converges to the same solution for all starting points, when a large number of starting points are used.

4.4 Questions

In order to estimate the usefulness of the model, we ask a number of questions which we will try to answer using numerical experiments.

As the current strategy of our partner company is similar to a fixed mix strategy, we would like to investigate if using dynamic asset allocation in the model will improve its performance. Although dynamic asset allocation have been found to

Case	Split	Length, months	Size (rows/cols/scenarios)
Case 1, 2	30*10*10	6,12,24	164179/197823/3000
Case 1, fix	30*10*10	6,12,24	180834/201154/3000
Case 3:1	10*8*8*8	6,12,12,12	294639/353883/5120
Case 3:2	10*6*4*4*4	6,6,6,12,12	264079/316463/3840
Case 3:3	8*6*4*4*4	6,6,6,12,12	211273/253183/3072
Case 4, fix	8*6*4*4*4	6,6,6,12,12	231718/257272/3072

Table 1: Scenario tree sizes.

outperform a fix-mix approach (See Kouwenberg [22] and Fleten, Høyland and Wallace [15]), we still perform these tests in order to make sure that this is true also for our chosen methods, as well as for establishing a reference value against which performance differences depending on scenario tree shape may be compared.

Other researchers (Høyland and Wallace [15], Kouwenberg [22], and Higle [18]) have made clear that the way scenario trees are generated is most important for the performance of the system. As the generation of scenario trees by fitting of the stochastic properties is rather expensive (creating a scenario tree takes significantly longer than solving the resulting problem), we would like to see if this difference is significant for our problem.

Finally we perform a study to determine if changing the shape of the scenario tree (making it longer but more narrow) will affect the performance.

4.4.1 Fix-mix versus SP

In order to determine if the stochastic programming solution fares better than the fix-mix approach we apply both methods to 150 scenarios 5 years in length, with the portfolio being re-balanced every 6 months. All scenario trees for this test is generated by fitting the statistical properties of the underlying distribution. We report results from these simulations in Table 3 as Case 1. We report the difference and the standard deviation of the difference, as a large portion of the variability in the solutions is explained by differences between the scenarios. The scenarios are generated using antithetic sampling, in order to reduce the variance of the sampled mean, and we hence report two values for the standard deviation: the value obtained from all samples, and the value obtained when we average over each antithetic pair before calculating the standard deviation. If we assume that the two methods of generating scenario trees are equivalent, each scenario (or pair of scenarios) is a sample of the random variable defined as the merit function difference of the two methods used on a random scenario. As the number of runs is fairly large, we may assume that the mean of these variables has a normal distribution with standard deviation

$$\sigma_{\text{mean}} = \frac{\sigma_{\text{sample}}}{\sqrt{n}}$$

with n being the number of samples. Observing that $0.8198/\sqrt{75} = 0.0947$ we see that the mean difference of the two methods equals approximately 1.6 times the standard deviation. More specifically we see that the probability of getting a larger deviation from 0 given that none exists is 10.8%, which is the strength of the test of our claim. The resulting asset fractions are reported in Table 2 together with their standard deviations (only the actively traded assets are reported.

	Swedish bonds	Foreign bonds	Swedish stock	Foreign stock	Swedish t-bill
Fix-mix	0.62/0.0	0.04/0.0	0.13/0.0	0.09/0.0	0.005/0.0
SLP	0.67/0.04	0.0/0.0	0.12/0.03	0.09/0.03	0.0/0.0

Table 2: Asset fractions owned in Case 1 (fraction/std.).

4.4.2 Random versus optimized scenarios

Partly because the price of constructing scenario trees by optimization is rather high (constructing the scenario tree takes significantly longer than solving the resulting SLP with our implementation), we wish to determine if constructing scenario trees by fitting the properties of the underlying random distribution will give a significantly better performance compared to constructing trees by anthitetic random sampling with adjusted means. We run 150 scenarios using both randomly generated scenario trees and fitted trees. The results from this test are reported in Table 3, as Case 2. Using the same methods as earlier, we see that the probability of getting a larger deviation from 0, given that no difference exists, is 97%, and we may hence not draw the conclusion that a difference exists between the methods. Hence the rest of the trials will be carried out using scenario trees generated by random antithetic sampling with correction.

If we look at the asset fractions obtained by the two methods, as reported in Table 4, we see (not surprisingly) that the first-stage solutions obtained from randomly generated trees have a larger standard deviation. More important is the fact that there exists a statistically significant bias: when using a random tree, a larger fraction of the wealth is invested in stock.

4.4.3 Tree size

In this study we have used rather wide and short trees compared to other studies, mainly to compensate for the larger number of assets. In order to see if this choice affects the performance of the method, we try other shapes of trees, which are more narrow and deep, as this kind of shape is more common in other researchers' tests. For instance, Dempster and Consigli [8], having 5 assets, uses a 10 stage tree divided as $7, 3, 2^7$ (the first stage splits into seven nodes, each splitting in three in the next stage, each splitting in 2 in the next seven stages). Kouwenberg [22],

Method	Mean (BSEK)	Std (BSEK)
Case 1		
SP	22.2606	4.0778/1.9555
Fix-mix	22.1085	3.3445/1.4798
Difference	0.1521	1.1525/0.8198
Case 2		
Fitted tree	22.2606	4.0778/1.9555
Random tree	22.2641	4.179/1.88719
Difference	-0.0035	1.2585/0.8981
Case 3		
30*10*10	22.2641	4.179/1.88719
10*8*8*8	22.3193	4.8865/2.1128
Difference	-0.0552	2.4121/1.8296
30*10*10	22.2641	4.179/1.88719
10*6*4*4*4	22.6856	4.5646/1.9961
Difference	-0.4215	2.1907/1.5004
30*10*10	22.2641	4.179/1.88719
8*6*4*4*4	22.5202	4.9785/2.1256
Difference	-0.2561	2.4331/1.5880
Case 4		
SP	22.5202	4.9785/2.1256
Fix-mix	22.1168	3.4093/1.5139
Difference	0.3489	2.8009/1.7033

Table 3: Merit function values.

	Swedish bonds	Foreign bonds	Swedish stock	Foreign stock	Swedish t-bill
rand. tree	0.64/0.08	0.005/0.02	0.13/0.5	0.11/0.05	0.0/0.0
opt. tree	0.67/0.04	0.0/0.0	0.12/0.03	0.09/0.03	0.0/0.0

Table 4: Asset fractions owned in Case 2 (fraction/std.).

having 4 assets, uses a configuration of $10, 6^2, 4^2$, and Høyland and Wallace [20], having 4 assets, uses a configuration of $60, 6^2$.

In order to test if changing the shape of the tree will affect the efficiency of our method, we test three sizes of trees. The trees of each size are all generated by antithetic random sampling with corrected means. The sizes of these trees are given in Table 1, as Case 3. The length of the stages in the trees are adjusted to give the tree the same overall length in time, as we otherwise may not rule out that a difference in performance stems from different time horizons. We use 3 different trees, one with five stages and two with six stages in addition to the base case with four stages. As may be seen from Table 3, the two longer trees used, having 6 stages, perform significantly better than the base case with 4 stages. Also well worth noting is that the trees with 6 stages both outperform the tree with 5 stages, despite having a lower total number of scenarios. If we perform the same statistical test as before, we see that the probability of getting a larger deviation from zero when comparing Case 3:3 to the base case is 16%, suggesting that using longer and deeper trees does actually improve performance. In this comparison the trees have essentially the same number of scenarios, 3000 vs. 3072, but the size of the deterministic equivalent problems differ, as a longer tree will give more variables and constraints. When comparing Case 3:1 and 3:2, the probability of obtaining a larger difference given that none exists is 8%, again suggesting that deeper and more narrow trees perform better. Worth noting in this case is that the longer tree has significantly fewer scenarios (3840 versus 5120), although the difference in the size of the deterministic equivalent is smaller.

The fractions of available assets invested into different asset classes at time-stage 0 is given in Table 5. As may be seen from this table, the longer but more narrow scenario trees yields solutions with a higher fraction of the wealth invested in stock. This means that aggregating time-stages in the scenario tree will give the solution a bias, and that this bias is large enough to make longer but more narrow trees outperform shorter and wider trees, despite a significantly larger instability in the solutions.

4.4.4 Fix-mix versus SP revisited

As the performance seems to improve when using deeper, more narrow trees, we again try the fix-mix version of our model, using the trees given in Case 3:3. This test is reported as Case 4 in Table 3. Here, the difference between the two methods is bigger compared to test Case 1, and the probability of getting such a big difference

	Swedish bonds	Foreign bonds	Swedish stock	Foreign stock	Swedish t-bill
base case	0.64/0.08	0.005/0.02	0.13/0.5	0.11/0.05	0.0/0.0
case 3:1	0.60/0.12	0.02/0.05	0.15/0.09	0.12/0.09	0.003/0.02
case 3:2	0.58/0.12	0.03/0.08	0.18/0.09	0.10/0.08	0.002/0.02
case 3:3	0.51/0.18	0.07/0.13	0.19/0.09	0.11/0.09	0.01/0.07

Table 5: Asset fractions owned (fraction/std.).

given that none exists is approximately 8%. This is not surprising, as being required to keep the same asset mix is more of a handicap in a deeper tree, which has a higher number of opportunities to re-balance the portfolio.

5 Conclusions

In this paper we have developed an ALM model for a Swedish life insurance company, including a model of the surrounding economy. Two procedures for constructing scenario trees from the economy model are implemented and tested. We perform rolling horizon simulations to compare the different scenario generation techniques, and compare if having different shapes of the scenario trees impacts the performance of the model, when applied to out of sample scenarios.

By comparing different shapes of the scenario tree, we see that trees with many stages but few branches at each stage clearly outperform shorter, wider trees, despite the fact that the solutions obtained using these trees are very unstable (the fractions invested in different asset classes vary significantly between different scenario trees). In fact, the gain in out of sample performance gained by going from a four stage scenario tree to a six stage scenario tree was significantly larger than the gain from going from a fixed-mix approach to a stochastic programming approach using four stages.

In this work we have used anthitetic random sampling with correction as the method to generate scenarios, as our tests indicated no difference in performance between the two scenario tree generation methods used. However, the solution obtained by using random-corrected trees not only had a larger variability in the solutions, these solutions also had a larger investment in stock than the solutions obtained using fitted trees. Hence random sampling will not only give the solutions a larger variability, it will introduce a bias as well. When comparing different shapes of the scenario tree, we found that long,narrow trees performed better, while investing even more in stock. Hence it seems as if the relatively good performance of the random sampling technique (compared to Kouwenbergs findings [22]) may be explained by the fact that the random sampling introduces a bias which counteracts the bias from using scenario trees with aggregated stages.

6 Acknowledgments

This work was done in cooperation with Nordea Life & Pension, who also partially funded the project.

References

[1] G. C. Boender, *A hybrid simulation/optimisation scenario model for asset/liability management*, European Journal of Operational Research, 99 (1997), pp. 126–135.

[2] S. P. Bradley and D. B. Crane, *A dynamic model for bond portfolio management*, Management Science, 19 (1972), pp. 139–151.

[3] M. J. Brennan and E. S. Schwartz, *Alternative investment strategies for the issuers of equity linked life insurance policies with an asset value guarantee*, The Journal of Business, 1 (1979), pp. 63–93.

[4] M. J. Brennan and E. S. Schwartz, *An equilibrium model of bond pricing and a test of market efficiency*, Journal of Financial and Quantitative Analysis, 17 (1982), pp. 301–329.

[5] D. R. Cariño, D. H. Myers, and W. T. Ziemba, *Concepts, technical issues, and uses of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 450–462.

[6] D. R. Cariño and W. T. Ziemba, *Formulation of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 433–449.

[7] D. R. Cariño, W. T. Ziemba, T. Kent, D. H. Myers, C. Stacey, M. Sylvanus, A. L. Turner, and K. Watanabe, *The Russell–Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming*, Interfaces, 24 (1994), pp. 29–49.

[8] G. Consigli and M. Dempster, *Dynamic stochastic programming for asset–liability management*, Annals of Operations Research, 81 (1998), pp. 131–161.

[9] J. Cox, J. Ingersoll, and S. Ross, *An intertemporal general equilibrium model of asset prices*, Econometrica, 53 (1985), pp. 363–384.

[10] J. Cox, J. Ingersoll, and S. Ross, *A theory of the term structure of interest rates*, Econometrica, 53 (1985), pp. 385–407.

[11] G. B. Dantzig, *Linear programming under uncertainty*, Management Science, 1 (1955), pp. 197–206.

[12] G. B. Dantzig and M. N. Thapa, *Linear Programming*, vol. 1, Springer-Verlag, New York, 1997.

[13] C. Dert, *Asset liability management for pension funds; A multistage chance constrained programming approach*, PhD thesis, Erasmus University Rotterdam, 1995.

[14] C. L. Dert, *A dynamic model for asset liability management for defined benefit pension funds*, in Worldwide Asset and Liability Modeling, W. T. Ziemba and J. M. Mulvey, eds., Cambridge University Press, 1998, ch. 20, pp. 501–536.

[15] S.-E. Fleten, K. Høyland, and S. W. Wallace, *The performance of stochastic dynamic and fixed mix portfolio models*, European Journal of Operational Research, 140 (2002), pp. 37–49.

[16] B. Golub, M. Holmer, R. McKendall, L. Pohlman, and S. Zenios, *A stochastic programming model for money management*, European Journal of Operational Research, 85 (1995), pp. 282–296.

[17] J. Gondzio and R. Kouwenberg, *High performance computing for asset liability management*, Operations Research, 49 (2001), pp. 879–891.

[18] J. L. Higle, *Variance reduction and objective function evaluation in stochastic linear programs*, INFORMS Journal on Computing, 10 (1998), pp. 236–247.

[19] K. Høyland, *Asset liability management for a life insurance company: A stochastic programming approach*, PhD thesis, Department of Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway, 1998.

[20] K. Høyland and S. W. Wallace, *Analyzing legal regulations in the Norwegian life insurance business using a multistage asset-liability management model*, European Journal of Operational Research, 134 (2001), pp. 293–308.

[21] J. Kallberg, W. R.B, and W. Ziemba, *Short term financial planning under uncertainty*, Management Science, 28 (1982), pp. 670–682.

[22] R. Kouwenberg, *Scenario generation and stochastic programming models for asset liability management*, European Journal of Operational Research, 134 (2001), pp. 279–292.

[23] M. Kusy and W. Ziemba, *A bank asset and liability management model*, Operations Research, 34 (1986), pp. 356–376.

[24] C. Maranas, C. Androulakis, C. Floudas, A. Berger, and J. Mulvey, *Solving long term financial planning problems via global optimization*, Journal of Economic Dynamics and Control, 21 (1997), pp. 1405–1425.

[25] G. Pflug, *Scenario tree generation for multiperiod financial optimization by optimal discretisation*, Mathematical Programming, Series B, 89 (2001), pp. 251–271.

[26] G. Pflug, A. Świętanowski, D. Dockner, and H. Moritch, *The AURORA financial management system: model and parallel implementation design*, Annals of Operations Research, 99 (2000), pp. 189–206.

[27] G. C. Pflug and A. Świętanowski, *Dynamic asset allocation under uncertainty for pension fund management*, Tech. Rep. TR1998–15, Vienna University, 1998.

A Bond pricing

In our version of the Brennan–Schwartz two-factor model, two state variables are used: the return on a console bond (a bond that never matures), and the instantaneous rate of return. The movements of these returns are given by the stochastic differential equation

$$\begin{aligned} dr =&\alpha_r(l - s - r)dt + \sigma_r r dz_r, \\ dl =&\alpha_l(\hat{l} - r)dt + \sigma_l l dz_l, \end{aligned}$$

where

r	instantaneous interest rate,
l	console rate,
$\alpha_{r,l}$	mean reversion strength for the two processes,
$\sigma_{r,l}$	standard deviation parameter,
$\bar{l}$	mean reversion level of console rate,
s	difference between long and short rate,
ρ	instantaneous correlation of dz_l and dz_r,
λ	market price of short-term interest rate risk,
c	continuous coupon payed by bond,
τ	time to maturity,
$B(l, r, c, \tau)$	price of bond as a function of the state variables and bond properties.

Bonds are priced by numerically solving the partial differential equation

$$\begin{aligned} &B_{rr}\sigma_r^2 r^2/2 + B_{rl}\rho\sigma_r r\sigma_l l + B_{ll}\sigma_l^2 l^2/2 + B_r(\alpha_r(l - s - r) - \lambda\sigma_r r) + \\ &B_l(\sigma_l^2 l\alpha_l^2(\hat{l} - l)^2 + l^2 - rl) - B_\tau + c - Br = 0, \end{aligned}$$

with the boundary condition $B(l, r, c, 0) = k$, where k is the face value of the bond (stating that the bond must yield its face value at maturity). For a derivation of this formula, see [4].

B Data

α_r	1.2492
α_l	0.1884
σ_r	0.1555
σ_l	0.1874
l	0.0523
s	0.0131
ρ	0.5808
λ	-0.4

Table 6: Parameter values for interest rate model.

	Risk premium	Standard deviation
Swedish bonds (SB)	-	0.0644
Swedish T-bills (ST)	-	0.0212
Swedish stock (SS)	0.07	0.2487
Foreign bonds (FB)	-	0.0951
Foreign stocks (FS)	0.06	0.1805
Swedish real estate (ES)	0.07	0.1805
Swedish real bonds (RB)	0.03	0.0355

Table 7: Means and standard deviations for asset model (yearly).

	SB	ST	SS	FB	FS	ES
ST	0.54					
SS	0.4643	0.1130				
FB	0.2600	0.1848	0.2313			
FS	0.3332	0.0641	0.6914	0.5801		
ES	0.6	0.4	0.2	0.1	0.05	
RB	0.2	0.1	0.2	0.2	0.1	0.2

Table 8: Correlations for asset model.

Paper II

Memory consumption versus computational time in nested Benders decomposition for stochastic linear programming

Fredrik Altenstedt
Department of Mathemathics
Chalmers University of Technology
412 96 Göteborg, Sweden

September 22, 2003

Abstract

The nested Benders decomposition algorithm is a popular algorithm for solving multi-stage stochastic programming problems. In this work we focus on using nested Benders decomposition to solve a large number of similar problems on smaller computers, a situation which arises when a stochastic programming based decision support system is to be tested. We present a number of techniques which may be used to trade memory requirements for execution speed, making it possible to tune the algorithm to the available memory. Finally we experiment on test-problems from both the literature and our own work. From these tests, we see that for the given test-problems, reductions in computational time on the order of 25% are possible, by using slightly more memory. Furthermore, we show that huge problems may be solved on commodity single processor computers, as long as care is taken to reduce the memory consumption.

Keywords: Stochastic linear programming; nested Benders decomposition; Optimization

1 Introduction and motivation

In recent years stochastic programming has become a popular tool for applications in which decisions must be made with limited information. After the initial decision is made, random information is revealed, and corrective actions may be taken. Such a sequence of action, random event and reaction may be repeated several times, resulting in a multi-stage stochastic programming problem. Optimization problems describing this sequence of decisions and random outcomes have applications in finance (see for example [7, 6, 8, 19, 14]) and power production ([21, 9, 18, 20]) among other fields. In stochastic programming, the future uncertain events are represented by scenarios, and the problem sizes grow exponentially in size with the number of time-stages considered in the model, possibly reaching tens or hundreds of millions of decision variables. In order to tackle such huge problems, decomposition methods have been developed, first and foremost the nested Benders decomposition method, where large problems are divided into several subproblems of lower dimension. As noted by some researchers, among others Birge et al. [4], the size of the problems that are possible to solve is limited partly by the amount of memory available. Hence, techniques by which to reduce the memory requirements of the nested Benders methods may increase the maximum size of problems that may be treated.

When we are solving a large number of rather small, similar problems, it might be worth the effort to try to reduce the computational time for each of these smaller problem, even though each individual problem takes a small amount of time to solve. In this setting, solving one of these problems several times with different solver settings to find a good setting for the problem-type may still pay off, as the improvements in solution efficiency is of benefit to all the problems. Testing and evaluating a decision support system based on stochastic programming requires the solution of a large number of similar stochastic programs. As examples of this we may note Kouwenberg [12], in which rolling horizon simulations are used, and Høyland [15] where the same testing technique is employed, using a large number of test-cases. Hence, an ability to trade memory requirements for execution speed in the nested Benders decomposition algorithm may be useful in two settings: firstly, when we wish to solve a large number of smaller problems, we may save computational time by actually using all the available computer memory. Secondly, if we are to solve larger problem instances, trading speed for lower memory requirements may make it possible to solve problems on computers with less memory, avoiding the need for special hardware.

The aim of our study is to implement and test different memory-reduction techniques on a number of test-problems from our own research and the literature, in order to determine which strategies provide a good tradeoff between memory requirements and computational time. Specifically, we look for strategies which are non-dominated in the sense that no other strategy consumes both less memory and time. In Section 2, we give a short description of the nested Benders algorithm and describe the difference between the implementation used in this work and the classic breadth-first implementation. The different techniques by which memory may be traded for speed are introduced in Section 3, and Section 4 describes our implementation of the nested Benders algorithm. The testing environment and the tests carried out are described in Section 5, and finally conclusions are drawn in Section 6.

2 Nested Benders decomposition

In this work, we use a version of the nested Benders algorithm in which the scenario tree is searched depth-first instead of the more commonly implemented breadth-first protocol. The rationale behind searching the tree depth-first is that this will allow us to use some memory saving techniques which are otherwise not possible. In order to clarify the difference between the two approaches, and to explain for which cases the methods are equivalent, we give a short description of the nested Benders algorithm. (For a more thorough description of the classical breadth-first nested Benders algorithm, see [5].)

When solving a multi-stage stochastic programming problem, we assume that all random outcomes are represented by a scenario tree. A scenario tree is a rooted tree in which a node represents a possible state of the world resulting from different possible random outcomes. The root node occurs with certainty, and represents what is known at time $t = 1$, before any randomness is revealed. The child-nodes of the root node represent the random outcome at time $t = 2$, and each of these node occurs with a given probability. Each of these nodes in turn have a set of possible outcomes occurring at time $t = 3$ and so forth, down to the leaves at the terminal time T. A possible realization of all the random outcomes defining a scenario is then a path from the root of the tree down to a leaf. In order to simplify notation, we will assume that all nodes have a set of children, although this set is empty for the leaf nodes. In the description below, we assume that all the nodes in the scenario tree are numbered (with the root node being node 1), that the operator $\mathrm{p}(i)$ returns the parent of node i and the set of children of the node i is given by $\mathbf{J}(i)$. We denote the (unconditional) probability of node i occurring by q_i. Further, we assume that the problem has T stages, we denote the set of time-stages by $\bar{\mathrm{T}} = \{1, \ldots, \mathrm{T}\}$, and the set of nodes belonging to time-stage t by $\mathrm{N}(t)$.

The nested Benders algorithm is applicable to problems of the form

$$\begin{aligned} \min \quad & \sum_{t=1}^{\mathrm{T}} \sum_{i \in \mathrm{N}(t)} \mathrm{q}_i {\mathrm{c}_i}^{\mathrm{T}} x_i, && \text{(1a)} \\ \text{s. t.} \quad & \mathbf{W}_1 x_1 = \mathrm{h}_1, && \text{(1b)} \\ & \mathbf{A}_i x_{\mathrm{p}(i)} + \mathbf{W}_i x_i = \mathrm{h}_i, \quad t \in \bar{\mathrm{T}} \setminus \{1\},\ i \in \mathrm{N}(t), && \text{(1c)} \\ & x_i \geq 0, \quad t \in \bar{\mathrm{T}},\ i \in \mathrm{N}(t). && \text{(1d)} \end{aligned}$$

We describe the multi-cut version of the algorithm only, as the single-cut and aggregated multi-cut versions form minor variations of this algorithm. During the execution of the algorithm, we will solve optimality problems of the form

$$\begin{aligned} & [\mathbf{P}(i)] \\ \min \quad & z_i = \mathrm{c}_i^{\mathrm{T}} x_i + \sum_{j \in \mathbf{J}(i)} \theta_j, && \text{(2a)} \\ \text{s. t.} \quad & \mathbf{W}_i x_i = \mathrm{h}_i - \mathbf{A}_i x_{\mathrm{p}(i)}, && \text{(2b)} \\ & \mathbf{D}_{i,j}^k x_i + \theta_j \geq \mathrm{d}_{i,j}^k, \quad k \in 1, \ldots, r_{i,j},\ j \in \mathbf{J}(i), && \text{(2c)} \\ & \mathbf{E}_{i,j}^k x_i \geq \mathrm{e}_{i,j}^k, \quad k \in 1, \ldots, s_{i,j},\ j \in \mathbf{J}(i), && \text{(2d)} \\ & x_i \geq 0, \quad t \in \bar{\mathrm{T}},\ i \in \mathbf{N}(t), && \text{(2e)} \end{aligned}$$

and feasibility problems of the form

$[\mathbf{F}(i)]$

$$\begin{array}{llr} \min & w_i = \bar{1}^{\mathrm{T}} u_i^+ + \bar{1}^{\mathrm{T}} u_i^-, & \text{(3a)} \\ \text{s. t.} & \mathbf{W}_i x_i + \mathbf{I} u_i^+ - \mathbf{I} u_i^- = \mathrm{h}_i - \mathbf{A}_i x_{\mathrm{p}(i)}, & \text{(3b)} \\ & \mathbf{E}_{i,j}^k x_i \geq \mathrm{e}_{i,j}^k, \quad k \in 1, \ldots, s_{i,j},\ j \in \mathbf{J}(i), & \text{(3c)} \\ & x_i \geq 0, \quad t \in \bar{\mathrm{T}},\ i \in \mathbf{N}(t). & \text{(3d)} \end{array}$$

We denote the dual variables corresponding to the constraints (2b) in problem $\mathbf{P}(i)$ by π_i and the dual variables corresponding to (3b) in $\mathbf{F}(i)$ by γ_i.

When solving the root node problem $\mathbf{P}(1)$, we replace $h_1 - \mathbf{A}_1 x_{p(1)}$ by h_1 in constraint (2b). When solving a leaf node problem $\mathbf{P}(i), i \in N(T)$ we omit the variables θ_j and the constraints (2c), (2d), and (3c). Furthermore, whenever we have $r_{i,j} = 0$ in (2c), (that is, when we have no optimality cuts) we set θ_j to 0.

We may now express the recursive version of the nested Benders algorithm as follows:

```
begin
  \* Initialize and call solve_sub for the root node *\
  for $t \in \bar{\mathrm{T}} \setminus \{\mathrm{T}\}, i \in \mathbf{N}(t), j \in \mathbf{J}(i)$ do
      $s_{i,j} := 0$
      $r_{i,j} := 0$
  end
  do
      feas := solve_sub(1)
  while feas AND $z_1 < ubd_1$
end
where
funct solve_sub($i$) $\equiv$
  ⌈ret := true
   solve $\mathbf{P}(i)$
   if $\mathbf{P}(i)$ is unfeasible
      ret := false
   else
        \* If the current node problem is feasible, solve all children*\
        for $j \in \mathbf{J}(i)$ do
            feas := solve_sub($j$)
            if feas AND $\theta_j < z_j$
               \* Add an optimality cut if the child problem is feasible*\
               $r_{i,j} := r_{i,j} + 1$
               $\mathbf{D}_{i,j}^{r_{i,j}} := \frac{\mathrm{q}_j}{\mathrm{q}_i} \pi_j^{\mathrm{T}} \mathbf{A}_j$
               $\mathrm{d}_{i,j}^{r_{i,j}} := \frac{\mathrm{q}_j}{\mathrm{q}_i} (z_j + \pi_j^{\mathrm{T}} \mathbf{A}_j x_i)$
            else
               \* Add a feasibility cut if the child problem is infeasible*\
               $s_{i,j} := s_{i,j} + 1$
               $\mathbf{E}_{i,j}^{s_{i,j}} := \gamma_j^{\mathrm{T}} \mathbf{A}_j$
               $\mathrm{e}_{i,j}^{s_{i,j}} := (w_j + \gamma_j^{\mathrm{T}} \mathbf{A}_j x_i)$
               \* Do not solve any more children if one child is infeasible*\
                break
            endif
        end
        $ubd_i = z_i + \sum_{j \in \mathbf{J}(i)} \frac{q_j}{q_i} ubd_j - \theta_j$
        solve $\mathbf{P}(i)$
    endif
    if $\mathbf{P}(i)$ is feasible
      ret := true
    else
        \* Solve the feasibility form of the node problem
         if the optimality form is not feasible.*\
        solve $\mathbf{F}(i)$
        ret := false
        $ubd_i := \infty$
    endif
    return ret⌋
```

As may be seen from the algorithm, it is equivalent to the classic breadth-first version of the nested Benders algorithm, as long as we do not encounter any infeasible node problems. When we arrive at an infeasible node problem, the algorithms will differ, as the classic version immediately will start a backward pass, whereas the recursive algorithm will continue down the tree into subtrees which are not descendants of the infeasible node.

3 Memory saving techniques

In this section, we describe the different choices that may be made in the nested Benders algorithm which affect both the amount of memory used and the computational time required. The choice between breadth-first and depth-first does not directly affect the memory and time consumption, but it is included in this section as the choice affects the applicability of a number of memory saving techniques.

3.1 Breadth-first versus depth-first search

In the classic nested Benders decomposition, the node problems of the scenario tree are solved using a breadth-first search (BF), one stage at a time. Instead we may opt to solve the problems in depth-first (DF) order if this may help us save computational time or memory.

Before a subproblem may be solved in the nested Benders algorithm, it has to be loaded into the LP-solver used. The computational time for loading may be reduced by utilizing components already in the memory. If a neighboring subproblem is already loaded into the solver, it may prove advantageous to use data from this problem in order to avoid recreating the desired subproblem from scratch. The benefit from using an old problem largely depends on how similar the two problems are. Note that not only the time for setting up the problem is reduced; if the two

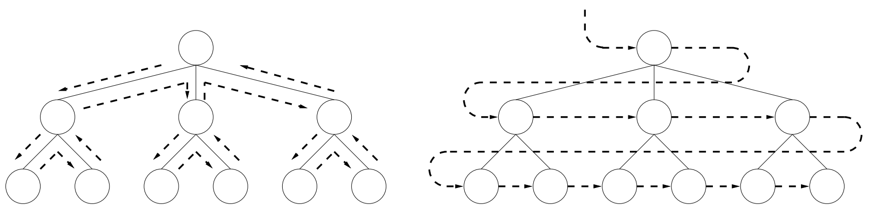
Figure 1: breadth-first versus depth-first.

problems are similar enough, a large portion of the (inverted) basis may still be valid, which may greatly reduce the solution time of the new problem. A complicating factor is the presence of optimality and feasibility cuts in the subproblems; these need to be stored and restored when a problem is loaded into the solver. The presence of cuts will hence reduce the benefits gained from using a neighboring problem.

If the problems $\mathbf{P}(i)$ are feasible for all nodes i, regardless of the values of $x_{\mathrm{p}(i)}$, then the problem is said to have *complete recourse*. If the optimization problem to be solved has this property, then no feasibility cuts will be generated during the run of the nested Benders algorithm. When we have complete recourse it is obvious that the two traversal methods (illustrated in Figure 1) are equivalent from a solution perspective, as each problem is solved twice in each iteration, once before and once after its child problems are solved. (The order in which the problems are solved will however differ between the two versions of the algorithm.) If a problem does not have complete recourse, a feasibility cut may be generated as the algorithm proceeds, whence the normal solution sequence will be interrupted. Since the order in which the node problems are solved differ between the two methods, this means that different nodes will already be solved when the interruption occurs; it shows that the two methods are not equivalent when feasibility cuts are generated.

Although the two methods are equivalent for problems with complete recourse when we only consider which nodes are solved, and in which relative order, the number of times each problem needs to be loaded into the LP-solver will differ between the two methods. When solving the whole problem using breadth-first search, we have, in essence, three choices of how many LP-problems to keep in working memory. We may choose to have one in total, one per level of the tree, or one per node of the tree. If we have one LP per node, it is obvious that each problem will be loaded into the LP solver only once. When fewer subproblems are used, then the number of times each problem is loaded will depend on the order in which the tree is searched. When a problem is loaded into memory, we differentiate between three different cases, as they consume different amounts of computational time:

1 The problem is constructed from scratch.

2 The problem is adapted from a neighboring problem, with no cuts being

	From scratch	From neighbor with cuts	From neighbor without cuts
BF 1 LP	$2q-2$	$2m-2(q-2)$	n_q-1
BF q LPs	0	$2m$	n_q
DF 1 LP	$1+2m+n_{q-1}$	0	n_q-n_{q-1}
DF q LPs	0	m	n_q

Table 1: Number of problem loading operations for each iteration except the first (the problem is assumed to have complete recourse).

added or stored.

3 The problem is adapted from a neighboring problem, with cuts being added and/or stored.

As an example of how may times these operations are performed, we assume that we have a scenario tree with q stages, and n_i nodes at stage i. In addition we define $m := \sum_{i=2}^{q-1} n_i$, that is, the number of nodes that are neither a root nor a leaf. For such a problem, the number of times each loading operation is carried out is given in Table 1.

As may be seen from Table 1, BF is clearly to prefer when we only use one LP. On the other hand, DF may be preferred if we use q problems, since each problem in the "middle" nodes (i.e., the nodes that are neither a root nor a leaf) is solved twice each time it is loaded, while these nodes have to be loaded every time they are to be solved with BF search.

3.2 Aggregation

As noted by Dempster and Thompson [11], aggregating stages of the tree to fewer but larger nodes may decrease the time spent solving a problem. Aggregating subproblems affect not only the solution time, but also the amount of memory required to solve a problem. Aggregation affects the memory consumed mainly in three ways:

- The size of the subproblems which are loaded into the LP-solver becomes larger.
- In the nested Benders algorithm, cuts are generated when information is transferred between stages. Fewer stages implies fewer stage boundaries, which implies fewer cuts.
- If we are not interested in anything but the first-stage solution, we may throw away the last stage solution after it has been used to generate a cut. Making the last stage longer, and thus larger, through aggregation will allow us to throw away more variables.

As some of these effects increase the memory consumption whereas others decrease it, the net effect will vary from case to case.

3.3 Blocking and bouncing

Having one LP for each node in memory obviously means that each LP needs to be loaded only once. As re-solving usually is faster that solving the same problem from scratch, having one LP for each node may lower the computational time required. However, for larger problems it is usually not possible to keep one LP for each node, due to memory constraints. As a compromise, we implement a way of sharing LPs in only a portion in the tree. We do this by having one LP for each node up to and including the *block-stage* and share LPs further down in the scenario tree (see Figure 2). As the vast number of nodes are located after the block-stage, sharing LPs in the nodes after the block-stage may save significant amounts of memory. If the strategy of sharing nodes after the block-stage is considered, then the computational price of adding a cut and re-solving will be greater after the block-stage than before it, as the problem has to be loaded into memory and any cuts stored restored. Hence we might wish to have as accurate a solution as possible available for the problems before the block-stage, when we solve the nodes after the block-stage. Accurate solutions may be achieved by dividing the nested Benders algorithm into major and minor iterations. A major iteration consists of a normal iteration, while a minor iteration is a normal iteration which stops at the block-stage. Minor iterations are repeated until no new cuts are generated at the stages before the block-stage, at which time a major iteration is performed. We call this strategy *bouncing* as the sequencing protocol bounces at the block-stage.

When the recursive version of the nested Benders algorithm is used, we may choose to have one LP for each node below the block-stage as well, LPs which are shared between subtrees below the block-stage. This setup is illustrated in Figure 2. As this figure illustrates we divide the scenario tree into blocks, and keep only one subtree block in memory at a time. If we share problems in this manner, it might be beneficial to bounce below the block-stage as well, as the effort of setting up the subproblem then might be discounted across several calls to the solver. In Figure 2 a minor iteration would consist of solving level 1, feeding the solution to level 2, solving level 2 and feeding cuts back to level 1. A major iteration with bouncing below the block-stage would consist of solving levels 1–3. When cuts are fed back to level 2 from level 3, level 2 is re-solved and the information fed back to level 3. The iteration bounces between level 2 and 3 until a termination criterion is met. Such criteria might be to run a fixed number of iterations, to run until no further cuts are generated in the subtree, or to run until the subtree has a specified tolerance between the upper and lower bound of the optimal value. Currently in our implementation, the user may only specify a number of times to bounce, but as a subtree is not re-solved unless there is a difference between the upper and lower bound of the objective function, setting the number of iterations high will cause the subtree to be completely solved. The algorithm will still converge, regardless of the bouncing scheme chosen. Convergence follows from the fact that there is only a finite number of cuts that may be generated by the algorithm (since each subproblem has only a finite number of optimal bases). As no cuts are deleted and the algorithm will either terminate or generate a new cut in each iteration, convergence follows.

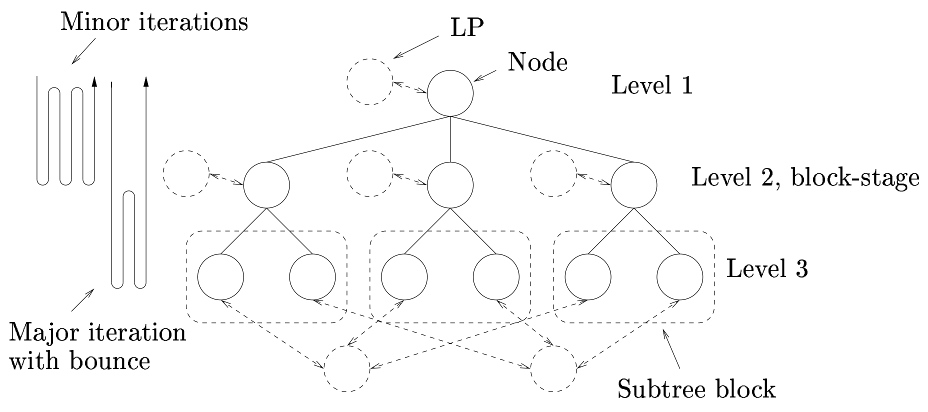

Figure 2: Sharing of LPs between subtrees and bouncing at the block-stage.

3.4 Basis compression

In order to be able to warm-start the solver when re-solving a subproblem, we need to store information about the optimal basis. As a variable or row may have a limited number of states in a basis (basic, non-basic lower bound, non-basic upper bound, etcetera) it is in our implementation enough to save 2 bits per row or column. Compressing the basis might seem an obvious improvement, but it may nevertheless save a significant amount of memory. (Using this technique for the PLTEXP5_16 test-problem reduces the cost of storing a full basis to 6.5 Mb compared to 105Mb for the format obtained directly from CPLEX.)

3.5 Purging cuts and bases

Usually, we store the optimal basis of a node problem as well as cuts generated between the times a subproblem is solved. If we need to greatly reduce the memory requirements we may purge optimal bases and cuts between major iterations. We illustrate this in Figure 3. Here, all cuts at level 3 and all optimal bases and variable values at level 3 and 4 are purged between major iterations. If we purge cuts and bases, convergence is no longer guaranteed, as we may end up in a cycle of regenerating cuts at level 3, while never generating cuts that are active in the optimal solution. To avoid cycling, each subtree must be solved to optimality in each major iteration if we are to purge cuts. It is clear that the algorithm will converge in this case, since solving each subtree to optimality in each major iteration is equivalent to aggregating levels 2–4, and solving the resulting two-stage problem by the nested Benders algorithm.

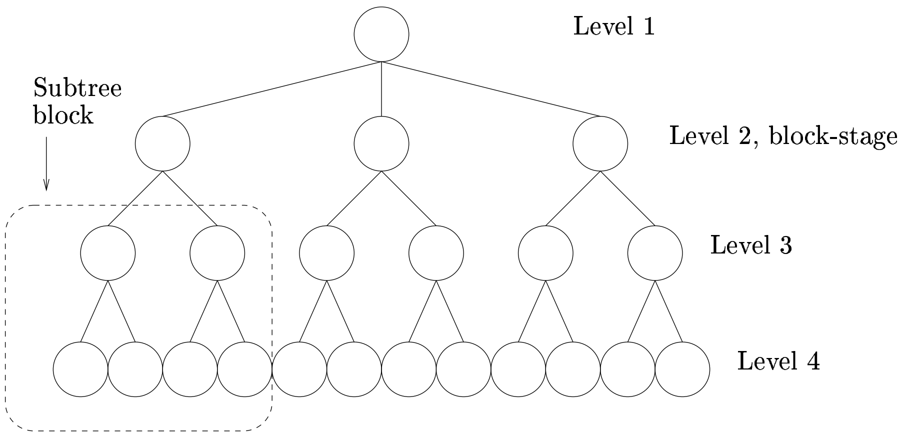

Figure 3: Purging data.

4 Implementation

As a part of our cooperation with a Swedish life insurance corporation, we have developed an asset liability management system to aid its decision making process. A part of the system consists of an implementation of the nested Benders algorithm, using either CPLEX, SOPLEX [24] or glpk [16] as the underlying LP-solver (in this work we have used CPLEX only). Currently the solver supports single-cut, multi-cut and aggregated multi-cuts similar to the ones described by Tsamasphyrou et al. [22] and Morton [17].

The only sequencing protocol implemented is Wittrock's fast-back–fast-forward [23], although it may be performed recursively or sequentially, with or without bouncing at a block-stage. In addition, aggregation is implemented, making it possible to aggregate an arbitrary number of subproblems anywhere in the tree; however, in this work we only aggregate the last stages in the scenario tree. A current weakness of the implementation is its poor performance when encountering infeasible subproblems. (When an infeasible subproblem is encountered, the problem (2) is converted into the problem (3), which is inefficient. Furthermore, the implementation does not permit us to retain the optimal basis of both the feasibility-problem and the optimality-problem, preventing us from doing a warm start for feasibility-problems.) As the primary objective of the solver is to be able to solve our model, described in [2], which has complete recourse, we have not put much effort into improving the solver for problems where feasibility-cuts are generated. We do not expect this poor performance to affect the results of our tests to a great degree, since only the problems in the WATSON test cases encounter infeasible subproblems during the solution, and for these problems, only a small number of infeasible node problems are found.

Another drawback of the current implementation that needs to be addressed is the storing of random data on scenario tree form. In the current implementation storing such random data requires unnecessary amounts of memory by using a slightly inefficient tree data-structure. If we succeed in removing the wasteful implementation of tree-structured data, the most probable effect would be to move the curves in Figures 4–7 to the left. As an example we may take the problems LIVIA_61440. In this problems approximately 8,800,000 random entries are stored using 250Mb. An efficient structure should use approximately 50Mb for the same data, shifting the graph of figure 7 200Mb to the left. Improving storage in this fashion should make the relative impact of using memory saving techniques larger.

5 Tests

We wish to determine which of the techniques described above may be used to affect the memory–time trade-off to benefit us most. In order to do so we have performed three rounds of tests. In the first set we apply a number of different configurations to all test problem, to find out which parameters might be interesting to vary. Second, we try different levels of aggregation on a smaller number of test-problems. Finally, we apply a large number of different solver configurations to a subset of the test-problems, in order to describe the memory–time trade-off in more detail.

5.1 Testing environment

All tests have been carried out on a Sun Blade 2000, having a 900Mhz SparcIII processor, with 1Gb of memory, and 8Mb of L2 cache. When running the tests, memory has been limited to 1Gb, that is no disk cache is allowed. The time required to read the problem is not included in the reported execution times, which are user times measured by the `getrusage` function. The only exception to this rule is the times reported in the aggregation tests. The times reported here are wall clock times from the function `gettimeofday`; the `getrusage` function does not have a high enough resolution, and is therefore not suited for this test. Throughout all the tests, we consistently use the multi-cut version of nested Benders decomposition.

5.2 Test-problems

The test-cases used in this work come both from our own research and from the literature. The families of test-problems used are:

- **LIVIA:** A multi-stage asset liability management model of a Swedish life insurance company, which is described by Altestedt [2]. All test-problems in this family have a scenario representation of the stochasticity, except the

Name	Stages	Scenarios	Rows	Columns
LIVIA_1920	5	1920	128749	154783
LIVIA_30000	5	30000	1676179	2012823
LIVIA_61440	10	61440	6745999	8007983
LIVIA_61440_b	10	61440	6745999	8007983
WATSON.256	10	256	43517	82177
WATSON.2688	10	2688	352013	671861
PLTEXP5_6	5	1296	161678	422876
PLTEXP5_16	5	65536	7270078	19014076
PLTEXP6_6	6	7776	970382	2537948
PLTEXP7_16	7	16777216	1861152446	4867629500
SGPF3y7	7	15625	761708	996117
SGPF5y7	7	15625	1230452	1543009
SFUND2560	6	2560	13629	52797
SFUND100000	6	100000	311109	1022217
SFUND100000_b	6	100000	311109	1022217

Table 2: Problem instances used in tests.

instance LIVIA_61440_b, which has a blocks representation (see Birge et al. [3] for a description of scenarios and blocks representation). As as consequence, the number of random parameters which need to be stored are much smaller in the latter case.

- **WATSON:** An asset liability management problem formulated by Dempster et al [10].
- **PLTEXP:** A stochastic capacity expansion problem. The biggest member of this family is only used as an illustrative example at the end of this paper.
- **SGPF:** A portfolio management problem. The problem is described by Frauendorfer et al. [13].
- **SFUND:** A simple portfolio management problem. This family of problems is described by Altenstedt [1]. The SFUND100000 and SFUND100000_b problems differ in the same way as the LIVIA problems above.

All these test-problems except LIVIA and SFUND are available via the web page http://www.stoprog.org/. The sizes of the test-problems used in this work are given in Table 2.

5.3 Initial tests

As previously mentioned, we first perform a preliminary test in order to determine which parameters and problems to experiment with further. In this test, we solve

case	common before cut	shared after cut	common last stage	bounce full	bounce leaf 1	aggregate 2	cut -1	cut -2	classic
1			x						
2									
3	x		x						
4			x			x			
5						x			
6	x		x			x			
7		x	x	x				x	
8		x		x				x	
9	x	x	x	x				x	
10			x	x		x	x		
11		x		x		x	x		
12	x		x	x		x	x		
13		x	x	x	x			x	
14		x		x	x			x	
15	x	x	x	x	x			x	
16			x	x	x	x	x		
17		x		x	x	x	x		
18	x		x	x	x	x	x		
19	x								x
20	x					x			x

Table 3: Parameter settings used in initial tests.

all the test-problems with 20 different settings of the solution parameters. These settings are summarized in Table 3, where an x in the table means that a specific option is used in that test. A block-level at -1 means that the block-level is the second to last level. In order to clarify all the options in Tables 3, we explain all the options for case 14. In this case, if the test-problem has 10 stages, then the cut is placed at stage 8. All nodes below the block-level use shared LPs, as described earlier. All nodes above the block-level uses one LP each. When solving the problem in case 14, we perform major iterations in which stages 1–10 are solved. In a major iteration, stages 9 and 10 are solved twice, as described in Section 3.3. Interspersed between the major iterations are minor iterations, in which levels 1–8 are solved to optimality.

The results from these initial tests are given in Appendix A, in Tables 5 and 6. As may be seen from Table 6, the computational times for PLTEXP vary significantly. The reason is that the node problems for the earlier stages have multiple optimal solutions, and the number of major iterations are strongly dependent on which solution is chosen during the first iteration of the algorithm. If we create a node problem from scratch and solve it, then we get a different optimal solution compared to taking an already solved neighboring problem, altering it and re-solving. Hence, the number of major iterations may vary in an unpredictable way, and the PLTEXP case is therefore not a good test-problem when we wish to test different strategies. Due to this variation in solution times for the PLTEXP set of problems, we will not use any problems from the PLTEXP test set for our further tests, except for a final illustrative example.

Furthermore, from Tables 5–6 we see that case 17 seems to be the option which minimizes the runtime for most cases (when the solution is not aborted due to violating the memory constraints). This leads us to believe that combining aggregation with sharing LPs between nodes in the later stages of the tree seems to be worth investigating further. As may be seen from Tables 5–6 almost no configurations for the smaller problems run out of memory, which makes us concentrate on the larger test-problems for our further testing. Hence, we will perform two new sets of tests:

- We test different levels of aggregation, combined with bouncing strategies for a small number of problems.
- We try a large number of different settings of parameters for the larger test-problems.

5.4 Aggregation

As may be seen from the preliminary tests, an aggregation of the two lowest stages in the scenario tree seems to shorten the solution time, as well as lower the memory required. We explore the effects of aggregation further by aggregating the last 2 to 5 stages, respectively, of the problems WATSON2668 and WATSON256, and aggregating the last 2 and 3 stages of LIVIA1920. These problems are chosen as they have a large number of stages, but different branching characteristics: the WATSON test-cases have a deeper and more narrow tree than the LIVIA test case. Rather small problems are chosen, as we want to be able to have one LP per node for all stages except the last stage. In these experiments, we use a common LP for the last (possibly aggregated) stage, while nodes not belonging to the last stage use one LP each. The results from these experiments are reported in Tables 7–12. (In these tests we measure the wall clock time since the system clock is not accurate enough, but as nothing except our experiments were run on the machine at the time, the times should be accurate enough for our purpose.) As may be seen in Tables 7–12, when looking at the aggregation strategy giving the shortest time, approximately 80–90% of the computing time is spent at the last stage. Although aggregating earlier stages may give lower computational times, we see that the potential gains from doing so is rather small. Aggregating earlier stages may however reduce the computational time spent in the last stage, as larger subproblems will provide more accurate solutions for the earlier stages. (More accurate solutions to the earlier stages result in more relevant cuts when these solutions are fed to the last stage, which possibly results in a lower number of iterations.) We explore the effect of providing the last stage with more accurate solutions by using the bouncing technique described earlier in Section 3.3. We see that using major and minor iterations (solving all but the last stage to optimality in between major iterations) makes the time spent solving the earlier stager higher, and the time spent in the last stage lower. As an example of this, we may compare Tables 11 and 8. Less time is spent in the last stage, and more in the earlier stages. As we might suspect, bouncing has a positive effect on the total computational time when a large number of stages are aggregated to a large final stage.

5.5 Memory versus computational time

In order to investigate the trade-off between memory and computational time further, we experiment with 7 of the test-problems. The test-problems we use are SFUND_100000 and SFUND_100000_b, LIVIA_61440 and LIVIA_61440_b, the two SGPF test cases and WATSON_2688.

In these tests, we will vary the following solver parameters:

- The last stage will consist of 1–4 aggregated stages (1 aggregated stage means no aggregation
- A block-stage, as described in Section 3.3, may be used; if so, it is placed at computational level −1, −2, −3 or −4 (−1 means at stage 5 if we have a 6-stage tree after aggregation), regardless of how may stages are aggregated.
- The nodes before the cut may have one LP instance per level, or one LP-instance each. If they use shared LPs then this will be marked with an 'x' in the column "common before cut" in Table 4.
- The nodes after the cut may have one LP instance per level, or share one instance between subtrees. A third combination used is to share LP-instances between subtrees, except for the last stage, which uses a common LP-instance. Which combination is used is indicated with an x in the columns "shared after cut" and "common last stage" in Table 4.
- The algorithm may bounce before the cut, solving all levels before the cut to optimality before solving the levels below the cut. This is indicated in the column "bounce full" in Table 4.
- The algorithm may bounce below the cut, solving everything below the cut twice before going back up the tree again. This is indicated in the column "bounce leaf 1" in Table 4.

In these tests we exclude all obviously redundant combinations (for example, if we have no cut, we do not need to test if bouncing at the cut-stage has any effect).

All possible non-redundant combinations are used for the WATSON and SGPF test cases. For the SFUND test-problems we restrict ourselves to aggregating the 2 or 3 last stages, and placing a cut at level −1,−2 or −3. For the LIVIA test-problems, we always aggregate 2, 3 or 4 stages, and place a cut at level −1, −2 or −3, respectively.

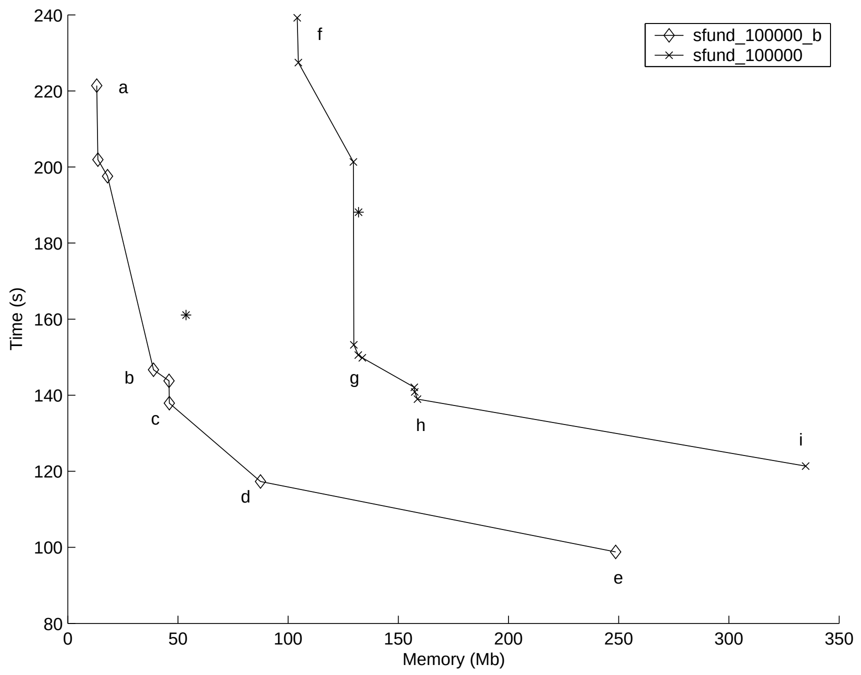

Figure 4: Memory versus time, SFUND_100000 and SFUND_100000_b.

The Figures 4–7 summarize these tests. The curves in these figures are formed by the runs which are not dominated by any other solution; this means that no other solver configuration has both lower memory consumption and shorter computational time. The results marked with '*' in these figures correspond to the results for the classic nested Benders decomposition algorithm, using breadth-first search and one problem per time-stage. For this method, the aggregation with the shortest solution time is used. The configurations of some of the points at the efficient frontier are given in Table 4.

5.6 A final illustrative example

As a final example, we solve the PLTEXP7_16 problem from the POST test set. In order to reduce memory consumption, the last two stages are aggregated, a cut stage is placed at level 3, and all data below the cut-stage is purged between the

case	common before cut	common after cut	common last stage	bounce full	bounce leaf 1	aggregation	cut-level	# major iter.	# minor iter.
WATSON_2688									
a	x	x	x			2	-4	6	13
b		x	x	x	x	3	-4	5	7
c				x	x	2	-4	6	12
d				x	x	2	-3	6	13
e				x	x	2	-1	5	16
f						2		9	0
SGPF3Y7									
a	x	x	x	x	x	2	-1	3	6
b	x			x		2	-1	3	5
c				x	x	2	-1	3	4
SGPF5Y7									
d	x	x	x	x	x	2	-1	2	3
e	x			x	x	2	-1	2	3
f				x	x	2	-1	2	3
SFUND_100000_b									
a	x	x	x	x	x	3	-1	18	44
b	x			x	x	3	-1	12	33
c				x	x	3	-1	15	32
d				x	x	2	-2	10	39
e				x	x	2	-1	12	55
SFUND_100000									
f	x	x	x	x	x	3	-1	13	51
g	x		x	x	x	2	-2	15	65
h			x	x	x	3	-1	13	56
i				x	x	2	-1	17	67
LIVIA_61440_b									
a	x	x	x	x		4	-3	17	69
b	x			x	x	4	-1	7	49
c	x			x	x	3	-2	7	49
LIVIA_61440									
d	x	x	x	x	x	4	-2	10	60
e	x			x	x	3	-2	9	72
f			x	x	x	3	-3	10	64

Table 4: A selection of non-dominated Test-cases from Figures 4–7.

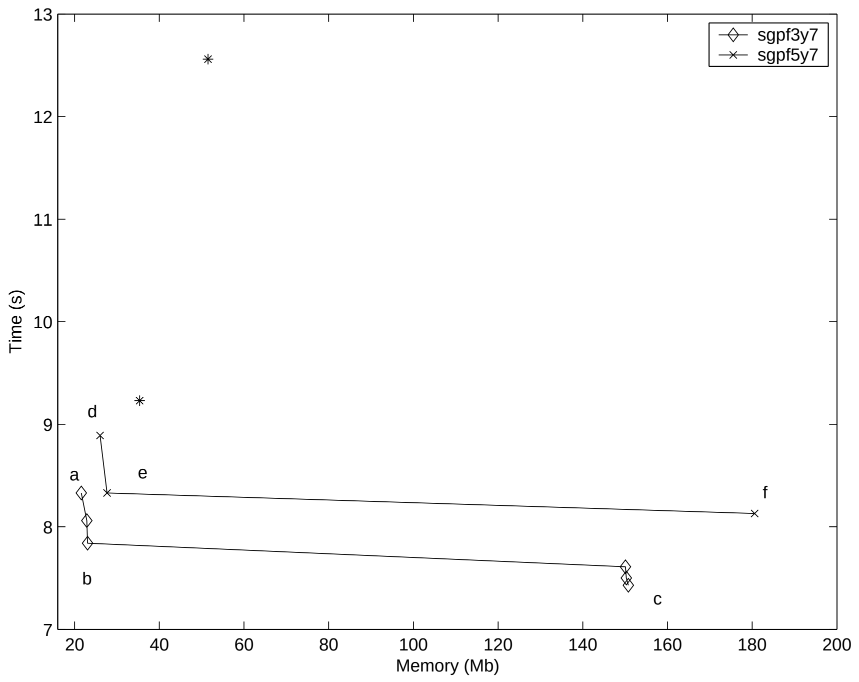

Figure 5: Memory versus time, SGPF.

major iterations. As data is purged, we need to completely solve the subproblem below the cut stage in every major iteration. Using this setup, the problem is solved in 7078 seconds using 809,280Kb of memory (giving the optimal value -32.821885). As noted earlier, the PLTEXP set of problems does not constitute a challenge for the solver, but this test problem is however illustrative from another point of view. According to Table 2, the problem has 1,861,152,446 rows and 4,867,629,500 columns. Only storing a primal and dual solution in double precision would require approximately 54Gb of memory. Being able to solve this problem using less than 1Gb of memory shows that large problems may be successfully solved on a single-processor computer, as long as care is taken to reduce resource consumption.

6 Conclusions and further research

We have explored a number of techniques which may be used to trade higher memory consumption for shorter execution times in the nested Benders algorithm. From the results of our tests, we believe that for small and medium size problems we may obtain significant gains in computational time at the price of a slightly higher memory consumption. These savings are of the magnitude of 25% compared

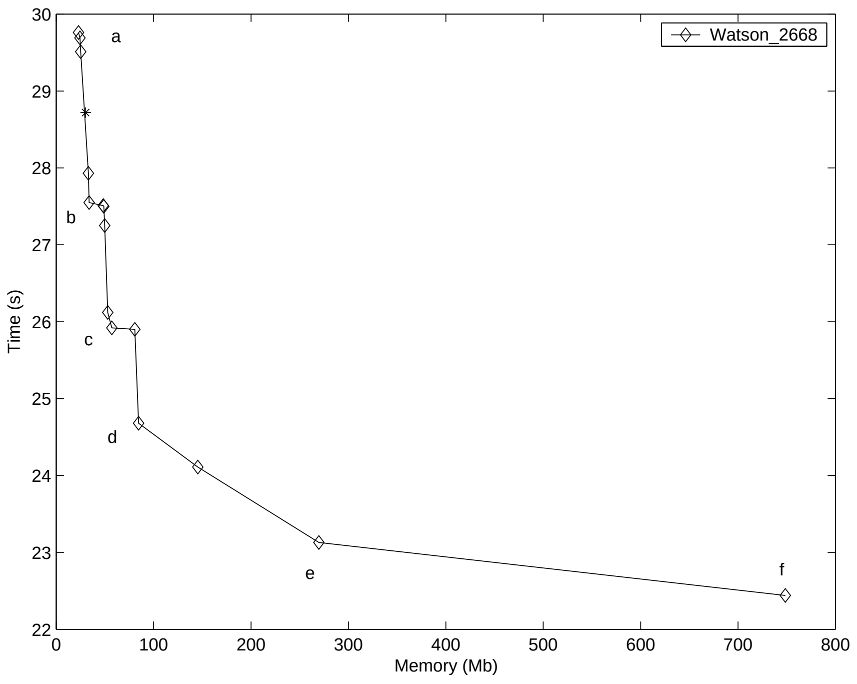

Figure 6: Memory versus time, WATSON_2688.

to the classic breadth-first implementation of the nested Benders decomposition algorithm, with optimal aggregation. The improvements which contribute to these shorter solution times is the sharing of LPs between nodes in the scenario tree, combined with different bouncing strategies which modify Wittrock's fast-back–fast-forward sequencing protocol. Furthermore, we have found that it is possible to solve large instances of SP-problems on limited hardware, by taking care to conserve computational resources.

In this work, the rules of when to perform major and minor iterations has been rather simple, and we would like to investigate these further, for instance using the difference between the upper and lower bounds at the levels after the block-level to determine if bouncing should occur. Furthermore, we would like to improve the performance of the solver when feasibility-cuts are generated. We believe that the poor performance when infeasible subproblems are encountered may be partially explained by the fact that we reuse the LP-structure of the optimality problem when constructing the feasibility-problem. If we were to associate a separate feasibility LP-instance to each node (for example sharing one feasibility LP between all nodes of a certain time-period), and store the last optimal basis of the feasibility-problem as well as the optimality-problem, we believe that this would improve performance for problems without complete recourse.

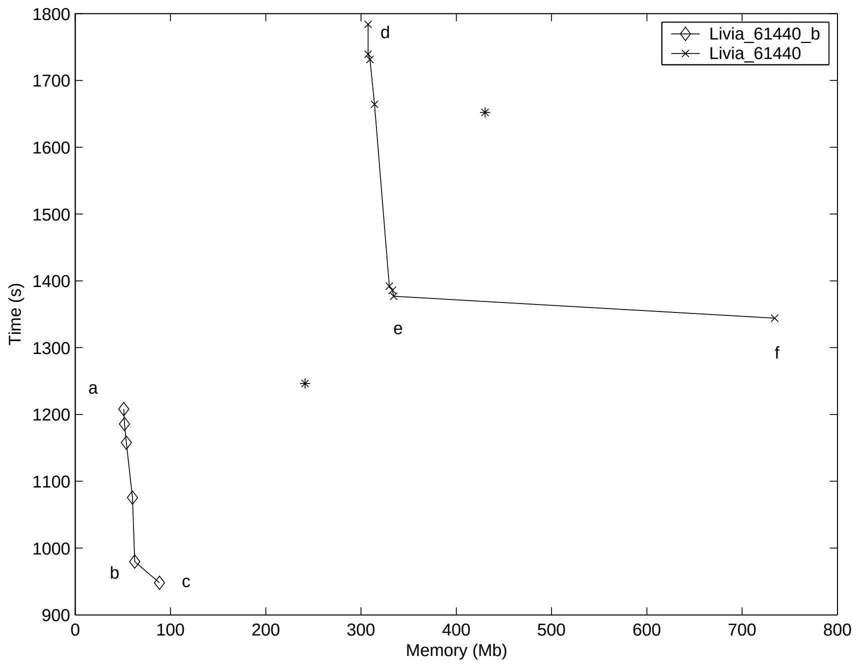

Figure 7: Memory versus time, LIVIA.

References

[1] F. ALTENSTEDT, *Asset aggregation in stochastic programming models for asset liability management*, Preprint 2003:48, Chalmers University of Technology, Department of mathematics, SE-412 96 Göteborg, Sweden, 2003. Submitted to Computational Optimization and Applications.

[2] ——, *An asset liability management system for a Swedish life insurance company*, Preprint 2003:47, Chalmers University of Technology, Department of mathematics, SE-412 96 Göteborg, Sweden, 2003. Submitted to Annals of Operations Research.

[3] J. BIRGE, M. DEMPSTER, H. GASSMAN, E. GUNN, A. KING, AND S. WALLACE, *A standard input format for multiperiod stochastic linear programs*, COAL Newsletter, 17 (1987), pp. 1–19.

[4] J. R. BIRGE, J. DONOHUE, CHRISTOPHER, D. F. HOLMES, AND O. G. SVINTSISKI, *A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs*, Mathematical Programming, 75 (1996), pp. 327–352.

[5] J. R. BIRGE AND F. LOUVEAUX, *Introduction to Stochastic Programming*, Springer-Verlag, Berlin, 1997.

[6] D. R. CARIÑO, D. H. MYERS, AND W. T. ZIEMBA, *Concepts, technical issues, and uses of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 450–462.

[7] D. R. CARIÑO AND W. T. ZIEMBA, *Formulation of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 433–449.

[8] D. R. CARIÑO, W. T. ZIEMBA, T. KENT, D. H. MYERS, C. STACEY, M. SYLVANUS, A. L. TURNER, AND K. WATANABE, *The Russell–Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming*, Interfaces, 24 (1994), pp. 29–49.

[9] C. CARØE AND R. SCHULTZ, *A two-stage stochastic program for unit commitment under uncertainty in a hydro-thermal system*, Tech. Rep. SC 98-13, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1998.

[10] G. CONSIGLI AND M. DEMPSTER, *Dynamic stochastic programming for asset–liability management*, Annals of Operations Research, 81 (1998), pp. 131–161.

[11] M. DEMPSTER AND R. THOMPSON, *Parallelization and aggregation of nested Benders decomposition*, Annals of Operations Research, 81 (1998), pp. 163–187.

[12] S.-E. FLETEN, K. HØYLAND, AND S. W. WALLACE, *The performance of stochastic dynamic and fixed mix portfolio models*, European Journal of Operational Research, 140 (2002), pp. 37–49.

[13] K. FRAUENDORFER, G. HAARBRÜCKNER, C. MAROHN, AND M. SCHÜRLE, *SGPF - portfolio test problems for stochastic multistage linear programming.* http://www.ifu.unisg.ch/sgpf/.

[14] B. GOLUB, M. HOLMER, R. MCKENDALL, L. POHLMAN, AND S. ZENIOS, *A stochastic programming model for money management*, European Journal of Operational Research, 85 (1995), pp. 282–296.

[15] R. KOUWENBERG, *Scenario generation and stochastic programming models for asset liability management*, European Journal of Operational Research, 134 (2001), pp. 279–292.

[16] A. MAKHORIN. Computer code available from http://www.gnu.org/software/glpk/glpk.html.

[17] D. P. MORTON, *An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling*, Annals of Operations Research, 64 (1996), pp. 211–235.

[18] M. P. Nowak, R. Schultz, and M. Westphalen, *Optimization of simultaneous power production and trading by stochastic integer programming*, Stochastic programming E-print series, (2002).

[19] C. Oguzsoy and S. Guven, *Bank asset and liability management under uncertainty*, European Journal of Operational Research, 102 (1997), pp. 575–600.

[20] M. Pereira and L. Pinto, *Stochastic optimization of a multireservoir hydroelectric system: A decomposition approach*, Water Resourses Research, 21 (1985), pp. 779–792.

[21] S. Takriti, J. Birge, and E. Long, *A stochastic model for the unit commitment problem*, IEEE Transactions on Power Systems, 11 (1996), pp. 1497–1508.

[22] P. Tsamasphyrou, A. Renaud, and P. Carpentier, *Transmission network planning under uncertainty with Benders decomposition*, in Optimization, Proceedings of the 9th Belgian–French–German Conference on Optimization, Namur, September 7–11, 1998, V. H. Nguyen, J.-J. Strodiot, and P. Tossings, eds., vol. 481 of Lecture Notes in Economics and Mathematical Systems, Berlin, 2000, Springer-Verlag, pp. 457–472.

[23] R. J.-B. Wets, *Stochastic programming models: wait-and-see versus here-and-now.* www.stoprog.org.

[24] R. Wunderling, *Paralleler und Objekt-orientierter Simplex-Algorithmus*, PhD thesis, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1996.

A Data

case	LIVIA_1920	LIVIA_3000	LIVIA61440	LIVIA61440_b	wat256	wat2688
	16.38	-	-	-	3.24	26.67
1	179736	>1e6	>1e6	>1e6	106856	576584
	12/0	-	-	-	8/0	10/0
	14.15	-	-	-	2.86	-
2	540464	>1e6	>1e6	>1e6	165408	>1e6
	12/0	-	-	-	8/0	-
	17.74	204.43	2240.50	1751.30	4.24	33.59
3	16512	156488	629552	440904	9184	36976
	12/0	12/0	26/0	21/0	9/0	10/0

case	LIVIA_1920	LIVIA_3000	LIVIA61440	LIVIA61440_b	wat256	wat2688
	12.05	210.01	-	-	2.75	25.15
4	52360	235616	>1e6	>1e6	56656	294264
	10/0	10/0	-	-	7/0	9/0
	10.34	-	-	-	2.53	22.96
5	230752	>1e6	>1e6	>1e6	126064	786416
	10/0	-	-	-	7/0	9/0
	13.37	222.58	1736.78	1319.17	3.48	29.78
6	14616	132264	500560	318408	8416	32576
	11/0	11/0	25/0	20/0	8/0	9/0
	18.95	194.70	-	-	3.97	31.56
7	56784	245552	>1e6	>1e6	58536	299840
	12/36	11/36	-	-	7/21	8/25
	19.78	234.07	-	-	3.95	31.93
8	60544	270896	>1e6	>1e6	58176	299496
	12/36	11/40	-	-	7/19	8/25
	19.42	199.33	5961.32	3823.69	5.46	53.46
9	15528	128256	699440	462248	8648	30440
	11/35	11/38	16/155	13/107	7/20	27/67
	11.14	191.71	-	-	3.11	25.86
10	50760	207024	>1e6	>1e6	56208	289528
	8/25	9/25	-	-	6/17	7/22
	11.37	193.92	-	-	3.16	26.22
11	50944	216112	>1e6	>1e6	55472	282688
	8/24	8/25	-	-	6/17	7/22
	13.53	198.69	4558.28	2691.99	4.72	38.47
12	12760	103168	547584	305920	7744	26376
	9/26	9/27	15/133	11/88	7/20	8/30
	15.28	177.08	-	-	3.66	26.85
13	54280	241232	>1e6	>1e6	58096	301248
	6/22	6/23	-	-	5/13	5/16
	14.39	178.97	-	-	3.46	25.41
14	57952	266928	>1e6	>1e6	57896	297192
	6/22	6/24	-	-	5/13	5/16
	17.01	178.93	3460.55	2401.90	4.98	38.10
15	15176	127472	593304	380872	8544	30344
	7/23	6/23	9/95	8/67	5/17	6/22

case	LIVIA_1920	LIVIA_3000	LIVIA61440	LIVIA61440_b	wat256	wat2688
	10.73	186.49	-	-	3.07	25.26
16	50008	207696	>1e6	>1e6	56768	292192
	5/17	5/19	-	-	5/14	5/17
	9.38	169.73	-	-	2.79	22.87
17	50072	216648	>1e6	>1e6	55488	283480
	5/17	5/19	-	-	5/13	5/16
	11.67	193.16	2676.46	1836.63	4.22	35.13
18	12432	102912	470096	255168	7712	26040
	5/18	6/18	8/80	8/57	5/16	6/27
	17.87	204.23	2115.02	1777.62	3.83	33.30
19	15416	153840	613328	434528	7704	33672
	12/0	12/0	25/0	22/0	8/0	10/0
	12.33	220.28	1653.34	1337.78	3.04	28.72
20	13864	131592	484944	312760	7392	30248
	10/0	11/0	25/0	21/0	7/0	9/0

Table 5: Solution times (s), memory required (Kbyte) and #major/minor iterations for test cases.

case	PLTEXP5_6	PLTEXP5_16	PLTEXP6_6	SGPF3y7	SGPF5y7	SFUND_2560	SFUND_100000
	1.98	-	13.05	17.66	-	9.98	-
1	81216	>1e6	481976	755224	>1e6	108312	>1e6
	1/0	-	1/0	6/0	-	17/0	-
	3.56	-	-	-	-	9.40	-
2	388992	>1e6	>1e6	>1e6	>1e6	264736	>1e6
	1/0	-	-	-	-	17/0	-
	4.78	1059.20	107.78	16.71	16.98	12.55	478.97
3	12488	407760	63456	52952	68648	13448	268328
	4/0	15/0	12/0	5/0	3/0	16/0	29/0
	1.00	33.21	4.85	8.70	11.43	5.77	157.23
4	22776	390688	116216	188344	234224	36912	365344
	1/0	1/0	1/0	5/0	4/0	15/0	27/0

case	PLTEXP5_6	PLTEXP5_16	PLTEXP6_6	SGPF3y7	SGPF5y7	SFUND_2560	SFUND_100000
5	2.85 214384 1/0	- >1e6 -	- >1e6 -	- >1e6 -	- >1e6 -	4.88 114096 16/0	- >1e6 -
6	3.06 11952 4/0	114.77 321728 4/0	21.06 48656 6/0	9.29 36664 5/0	12.16 54048 4/0	7.43 10864 17/0	190.74 140672 29/0
7	1.60 19632 1/0	83.52 227728 1/0	9.70 93496 1/0	14.62 181696 4/6	17.90 235232 3/3	11.66 45568 10/60	337.23 480680 18/104
8	1.05 28760 1/0	30.72 293032 1/0	5.08 103032 1/0	13.96 182752 3/4	20.10 241312 3/5	11.93 46920 10/60	355.26 485968 18/104
9	1.59 9472 2/2	153.01 155440 2/2	16.78 27280 2/3	15.17 38088 4/6	18.82 48368 3/5	16.97 14992 13/64	413.85 241576 17/108
10	0.99 18336 1/0	32.63 188056 1/0	4.73 89216 1/0	8.39 172616 4/5	10.19 213816 4/5	6.00 39288 9/49	130.66 351440 17/83
11	1.00 23560 1/0	29.34 222920 1/0	3.87 99000 1/0	7.18 169672 3/4	11.61 220056 3/4	6.35 39904 9/50	139.73 350432 16/91
12	1.69 7400 2/2	60.67 122688 2/2	8.54 21064 2/3	8.94 23664 4/5	10.88 32968 4/5	11.51 11872 11/60	217.25 140472 20/104
13	1.50 19632 1/0	83.72 227728 1/0	9.77 93496 1/0	16.46 181320 4/6	15.34 212672 2/2	9.49 41864 7/40	295.08 458728 14/64
14	0.96 28760 1/0	30.51 293032 1/0	5.09 103032 1/0	15.37 182872 3/4	15.04 217168 2/3	9.08 42800 7/40	290.88 465040 14/64
15	2.57 9472 2/2	153.34 155440 2/2	16.79 27280 2/3	17.07 38120 4/6	15.67 47008 2/3	13.71 14224 7/47	380.61 245000 15/74
16	0.99 18336 1/0	32.40 188056 1/0	4.74 89216 1/0	7.44 169224 3/4	8.41 205216 2/2	5.16 37200 6/34	121.48 349688 12/65

Stage	no aggr.	aggr. 2	aggr. 3	aggr. 4	aggr. 5
1	0.0038	0.0038	0.0052	0.0042	0.0039
2	0.0097	0.0094	0.0089	0.0072	0.0067
3	0.0203	0.0186	0.0174	0.0140	0.0139
4	0.0476	0.0363	0.0333	0.0262	0.0213
5	0.0909	0.0765	0.0646	0.0514	0.0584
6	0.1695	0.1641	0.1287	0.1083	4.8845
7	0.3449	0.3366	0.2555	3.3649	
8	0.7037	0.6606	2.7327		
9	1.2956	2.1971			
10	1.5501				
tot	4.2362	3.5030	3.2463	3.5763	4.9888
# iter	9	8	7	7	6

Table 7: Effect of aggregation, WATSON 256. Computational time (s) spent at different stages.

case	PLTEXP5_6	PLTEXP5_16	PLTEXP6_6	SGPF3y7	SGPF5y7	SFUND_2560	SFUND_100000
	1.01	29.14	4.00	7.30	8.04	4.80	120.58
17	23560	222920	99000	169504	204456	37248	343848
	1/0	1/0	1/0	3/4	2/3	6/32	17/67
	1.66	61.02	8.51	8.21	8.85	9.24	184.79
18	7400	122688	21064	23296	31504	11168	137592
	2/2	2/2	2/3	3/6	2/3	7/45	14/71
	1.45	80.72	9.06	24.04	15.24	12.81	460.92
19	11688	344424	50224	50520	66240	12536	245368
	1/0	1/0	1/0	6/0	3/0	17/0	26/0
	0.88	32.18	4.31	9.23	12.56	6.83	184.61
20	11568	317048	47016	35368	51496	9832	131632
	1/0	1/0	1/0	4/0	4/0	16/0	25/0

Table 6: Solution times (s), memory required (Kbyte) and #major/minor iterations for test cases.

Stage	no aggr.	aggr. 2	aggr. 3	aggr. 4	aggr. 5
1	0.0073	0.0068	0.0153	0.0094	0.0068
2	0.0391	0.0389	0.0514	0.0371	0.0301
3	0.1104	0.1093	0.1053	0.0896	0.0739
4	0.2336	0.2191	0.2066	0.1687	0.1541
5	0.5047	0.4576	0.4541	0.3301	0.4155
6	0.9963	0.9405	0.7958	0.7415	45.5003
7	1.9994	1.9685	1.4979	33.3550	
8	3.9859	4.0887	26.5642		
9	7.8155	21.9565			
10	16.8250				
tot	32.5171	29.7859	29.6907	34.7315	46.1807
# iter (maj).	10	9	10	8	8

Table 8: Effect of aggregation, WATSON 2668. Computational time (s) spent at different stages.

Stage	no aggr.	aggr. 2	aggr. 3
1	0.0395	0.0337	0.0213
2	0.3883	0.3391	0.3065
3	1.7977	1.5464	19.6230
4	5.9763	11.5137	
5	9.4417		
tot	17.6438	13.432955	19.9507
# iter (maj.)	12	11	8

Table 9: Effect of aggregation, LIVIA 1920. Computational time (s) spent at different stages.

Stage	no aggr.	aggr. 2	aggr. 3	aggr. 4	aggr. 5
1	0.7669	0.0081	0.0090	0.0063	0.0054
2	0.0258	0.0241	0.0164	0.0104	0.0104
3	0.0506	0.0475	0.0306	0.0190	0.0146
4	0.0935	0.0892	0.0522	0.0321	0.0232
5	0.1917	0.1749	0.0940	0.0587	0.0692
6	0.3676	0.3383	0.1621	0.1212	4.8555
7	0.7064	0.6508	0.2909	3.3151	
8	1.3668	1.2362	2.4486		
9	2.4143	1.8452			
10	1.0728				
tot	7.0564	4.4144	3.1038	3.5627	4.9784
# iter (maj./min.)	6/20	7/20	6/14	7/10	6/7

Table 10: Effect of aggregation, WATSON 256, bounce. Computational time (s) spent at different stages.

Stage	no aggr.	agg 2	aggr. 3	aggr. 4	aggr. 5
1	0.0266	0.0196	0.0296	0.0165	0.0132
2	0.1237	0.1279	0.0845	0.0559	0.0399
3	0.3277	0.3290	0.1769	0.1263	0.0840
4	0.6593	0.6235	0.3009	0.2172	0.1530
5	1.3285	1.2384	0.5555	0.4155	0.4437
6	2.5966	2.4228	0.9975	0.8226	44.1436
7	5.0231	4.5847	1.7887	30.8032	
8	9.6266	8.6005	22.2385		
9	17.8614	18.2199			
10	13.5898				
tot	51.1634	36.1664	26.1722	32.4571	44.8775
# iter (maj./min.)	11/39	8/30	8/18	7/16	7/11

Table 11: Effect of aggregation, WATSON 2668, bounce. Computational time (s) spent at different stages.

Stage	no aggr.	aggr. 2	aggr. 3
1	0.2726	0.1296	0.0487
2	1.8902	0.9544	0.3951
3	5.6822	2.5367	18.2527
4	13.0135	9.8344	
5	7.9559		
tot	28.8144	13.455069	18.696416
# iter (maj.)	10/38	9/26	8/13

Table 12: Effect of aggregation, LIVIA 1920, bouncing. Computational time (s) spent at different stages.

Paper III

Asset Aggregation in Stochastic Programming Models for Asset Liability Management

Fredrik Altenstedt
Department of Mathemathics
Chalmers University of Technology
412 96 Göteborg, Sweden

September 19, 2003

Abstract

Multi-stage stochastic linear programming is emerging as a valuable tool for addressing asset liability management problems (ALM problems). A number of researchers have demonstrated that the method used to generate the asset return scenarios used in the optimization has a major impact on the overall performance of the model. Less effort has been spent on trying to determine which assets should be used in the model. We explore the option of constraining the ALM model to use only a few linear combinations of the assets at hand, and to find the best linear combinations by optimization. Our hypothesis is that using such linear combinations may increase the performance of the model when the solution found is applied to out-of-sample scenarios. We believe our approach may be beneficial, as stochastic programming solutions have a tendency to chase spurious profits present only in the scenario tree; using artificial asset combinations should reduce this tendency, since fewer assets imply fewer spurious profit opportunities when the size of the scenario tree is held constant. We perform a number of tests on a simplified ALM problem, which show that restricting ourselves to use only a few linear combinations of assets does indeed improve performance. In fact, there is no statistically significant benefit of using more than two asset combinations for the model tested. This is the lowest number of synthetic assets we may have, while still retaining the ability to change the risk-reward trade-off.

Keywords: Stochastic programming; Asset liability management; Optimization; Finance.

1 Introduction

1.1 Background

When managing financial assets, the obvious goal of a manager is to maximize the yield of the funds invested, while keeping the investment risk low. Usually, the measure of risk is assumed to be static, that is, the goal of the investments does not change over time. However, for a large subset of investment problems, the yield of the investments is used to cover some form of liabilities. When this is the case, advantages may be gained by managing assets and liabilities in a coordinated fashion, resulting in an *asset liability management* (ALM) problem. For instance, the management of pension funds, where contributions from the fund's sponsor and yields from investments are supposed to cover future retirement liabilities, may be formulated as ALM problems. As the liabilities are dependent on the future inflation and wages of the beneficiaries of the fund, the target of the fund is uncertain and correlated with the development of the assets in which the fund invests.

Lately, stochastic programming has become a popular tool to tackle ALM problems. Stochastic programming solutions are dynamic in the sense that they explicitly account for the fact that the investment policy of the company will change when circumstances change. Taking these dynamics into account makes it possible to outperform static methods, such as the Markowitz portfolio optimization technique (which assumes that the investment policy will not change during the horizon considered). In the literature there are several examples of stochastic programming models used for asset liability management. For instance, Bradley and Crane [3] describe a model used for managing bond portfolios as early as 1972; other models are described in Cariño et al. [5, 4, 6], who constructed an ALM model to aid a Japanese insurance company, and Dempster and Consigli [7], who applied multi-stage SP to the problem of administering assets for a retirement fund. Dert [9] and Kouwenberg [17] have done the same in a Dutch setting, whereas Høyland [13] have treated the case of pension insurance, whose ALM problem has a slightly different characteristic. Other applications include index tracking, which have been addressed by Zenios et al. [21], while Mulvey and Vladimirou [18] treat more general asset management problems.

1.2 Motivation

In all the models mentioned above, except the one in Bradley and Crane [3] and Zenios et al. [21], stochastic programming is used for strategic asset allocation, where available funds are allocated between broad categories of assets, categories such as stocks, bonds, real estate and treasury bills. In these models the assets themselves are considered to be given; how they are selected is however not apparent. In this paper we will explore whether these classes of assets should directly be used in the stochastic programming model, or if benefits may be obtained by artificially creating other classes of assets, in effect reducing the space of possible investment strategies.

This type of approach has been investigated by Gaivoronski and de Lange [11]. They use an ordinary multi-stage stochastic programming model, with the addition that free trade is only allowed at stages 0 and 1. For all stages after stage 1 the fractions of wealth invested in different assets is required to be the same as in stage 1, giving a fix-mix strategy (with different asset proportions in different parts of the tree) According to their work, locking the assets in this fashion gave a better overall performance, compared to fully dynamic stochastic programming. Well worth noting is that when Gaivoronski and de Lange optimize the dynamic/fix-mix hybrid policy, they do this using a larger scenario tree than the one used for the fully dynamic solution. The reason for using different scenario trees is that the fully dynamic model is more computationally challenging, and the authors try to determine which approach is the best when the limiting factor is the available computational time. The main difference between our work and the work by Gaivoronski and de Lange is that they try to determine if fully dynamic solutions are worth the computational effort, whereas we try to determine if the fully dynamic solution may be improved by creating artificial constraints. We feel that the solution may be improved, as too many degrees of freedom in a stochastic programming model may cause the solution to adapt to peculiarities in the scenario tree representing the underlying distribution, rather than to the general properties of the underlying random distribution the scenario tree is supposed to reflect.

In order to carry out our investigations into the asset aggregation problem, we construct a bare-bones ALM problem. The problem is designed to be as simple as possible while still retaining the basic property of an asset liability problem; the goal is to invest means which are used to honor uncertain liabilities. Our interest in this problem stems from our wish to simplify our model of a Swedish life insurance company (described in Altenstedt [1]), but for simplicity we abstain from using the full model, as its complexity might hide the question we wish to treat.

1.3 Preview

In Section 2 we give a general description of how an ALM problem is structured. In Section 3 we classify the different simplifications that need to be made in order to make the general ALM-problem computationally tractable, and motivate why sometimes further reducing the already simplified problem might give solutions that are closer to the optimum of the unreduced problem. Two of the simplifications, namely the discretization of the distribution and the aggregation of assets, are discussed in Sections 3.4 and 3.5, respectively. We give a description of the simple ALM problem used in our tests in Section 4. In Section 5 we describe the tests performed and conclusions drawn in Section 6.

2 The canonical ALM problem

As mentioned above in Section 1.1, ALM models are designed to address the problem where funds are to be invested to cover future liabilities. In reality, there exist thousands upon thousands of possible assets between which an investor may divide their means, the price of which may change at any time. In addition, trading may be performed at each instance in time (with restrictions due to the closing of markets, etcetera). In order to simplify the formulation, we assume that decisions may only be taken at a discrete and finite set of times. As this set may be arbitrarily large we do not lose any significant generality. We further assume that the asset prices and other exogenous data such as liabilities are not influenced by the actions taken by the fund. Using these assumptions, we may formulate the full ALM problem faced by the fund managers as

$$\begin{aligned}
\min_{x_0} \quad & f_0(x_0) + \mathsf{E}^P_{\xi_1}[\min_{x_1(\xi_1)} f_1(\overrightarrow{x}_1(\xi_1), \xi_1) + \mathsf{E}^P_{\xi_2|\xi_1}[\min_{x_2(\xi_2)} f_2(\overrightarrow{x}_2(\xi_2), \xi_2) \\
& + \cdots + \mathsf{E}^P_{\xi_T|\xi_{T-1}}[\min_{x_T(\xi_T)} f_T(\overrightarrow{x}_T(\xi_T), \xi_T)] \cdots]], && \text{(1a)} \\
\text{s. t.} \quad & x_t(\xi_t) \in X_t(\overrightarrow{x}_{t-1}(\xi_{t-1}), \xi_t), \qquad t = 1, \ldots, T. && \text{(1b)}
\end{aligned}$$

where we introduce the following notation:

f_t	objective at time t (typically including penalties for undesirable conditions, and rewards for assets owned),
T	number of time-stages,
ξ_t	all random information available at time t,
$x_t(\xi_t)$	decision variables at time t (typically what is bought, sold and owned),
$\overrightarrow{x}_t(\xi_t)$	all decisions made up to and including time t, $\overrightarrow{x}_t(\xi_t) = (x_0, x_1(\xi_1) \ldots, x_t(\xi_t))$,
$X_t(\overrightarrow{x}_{t-1}(\xi_{t-1}), \xi_t)$	feasible set at time t dependent on random information and earlier decisions,
P	probability measure giving the probability of different outcomes.

In this formulation we have assumed that the decision at time t is taken directly after the random information at this time becomes known. All random variables in the problem are defined on a standard probability space, $(\Omega, \mathscr{F}, P)$. Here, Ω is the space of all possible outcomes, $\mathscr{F}$ is the algebra of all possible events on this space, generated by all possible outcomes of ξ^T, and P is a probability measure, giving a probability for all events in $\mathscr{F}$.

3 Reducing the full problem

3.1 Introduction

The full problem described above need to be further reduced in order to make it computationally tractable. The reduction of the problem may be done in at least four different ways:

1: The number of points in time when decisions can be made can be reduced, for instance by allowing the fund manager to change investments and other decisions fewer times per year. This category of simplifications also include reducing the total length of the problem by reducing the horizon.

2: The objective function may be changed, in order to make it easier to evaluate. For instance, a convex function may be replaced by a piecewise linear convex approximation.

3: The number of assets in which the fund may invest may be reduced, giving a lower number of variables. Removing assets from consideration may be seen as restricting the maximum owned amount to 0 for the removed assets. Hence removing assets simply makes the feasible set of the problem smaller. In the same fashion, aggregating several assets into wider aggregate assets, such as a stock index, may be accomplished by adding linear constraints to the model, fixing the proportions of different types of assets owned to predetermined values. We will refer to aggregating assets in this fashion as creating a *synthetic asset*, as the linear combination of a number of assets might be seen as yet another asset, which is bought and sold as an ordinary asset. Again, this aggregation results in a reduction of the feasible set.

4: The random distribution of possible outcomes may be simplified. Typically, this is done by discretizing the space of possible outcomes, reducing the possible outcomes to a scenario tree. As the probabilities of different possible outcomes are given by the probability measure P, discretizing the space of possible outcomes is the same as changing the measure P to a measure $\tilde{P}$, where the support of $\tilde{P}$ consists of finitely many outcomes.

In this work we concentrate on changes of type 3 and 4, and refer to a problem instance by the pair $\{X, P\}$. Here, X is the feasible set and P the probability measure defined over the space of possible outcomes Ω. Hence, changes of type 3 and 4 will affect X and P respectively. We illustrate the importance of looking at the two restrictions in tandem with a small example.

Consider a one stage investment problem, where the random yields are discretized using a number of scenarios (we assume that the actual distribution of the yields is known). In this problem there are many different stocks and bonds available to an investor. If the set of scenarios is not good enough to accurately describe all possible outcomes of the investments, there will exist investment opportunities that seem good under the probability measure defined by the scenarios ($\tilde{P}$), while performing worse under the true probability measure (P). If the investor exploits these opportunities, he/she might end up with an investment portfolio which performs well under the measure $\tilde{P}$, but badly under the measure P. If we restrict trade only to an index of bonds and another index of stocks, the investor no longer has an opportunity to exploit these false investment opportunities, and will choose a portfolio which will perform worse under the assumed probability measure $\tilde{P}$, but better under the real probability measure P.

3.2 Errors for two stage problems

If we initially restrict ourselves to two-stage versions of the problem (1), we may define the value of the recourse problem as

$$g(x_0, \xi) := \underset{x_1 \in X_1(x_0,\xi)}{\text{minimum}} f_1(x_0, x_1, \xi). \tag{2}$$

In this section we confine ourselves to make restrictions of the feasible set only at stage 0. Hence, the value of the recourse problem is not dependent on which feasible set is chosen. Using (2) we may define the objective function in the deterministic equivalent problem as

$$F(x_0, P) := f_0(x_0) + \mathsf{E}_\xi^P [g(x_0, \xi)] \tag{3}$$

We further define an optimal stage-0 solution when using the feasible set X and the measure P as

$$x_0^*(X, P) \in \underset{x_0 \in X}{\operatorname{argmin}} F(x_0, P). \tag{4}$$

For simplicity we assume that this solution is uniquely chosen whenever the optimal set is not a singleton.

In order to quantify the errors arising from changes of type 3 and 4 we introduce the notions of *restriction error* and *discretization error*. The discretization error is taken from Pflug [19], and it is defined as

$$d_X(P, \tilde{P}) := F(x_0^*(X, \tilde{P}), P) - F(x_0^*(X, P), P).$$

The value $d_X(P, \tilde{P})$ describes the loss from using the first-stage solution obtained by using the probability distribution $\tilde{P}$ instead of the optimal first-stage solution. (This corresponds to a change of type 4 above.) Note that the discretization error is dependent on which feasible set is used.

The restriction error gives the loss from restricting the first-stage feasible set from X to $\tilde{X} \subset X$ (a change of type 3 above). It is defined as

$$r_P(X, \tilde{X}) := F(x_0^*(\tilde{X}, P), P) - F(x_0^*(X, P), P);$$

by optimality, we know that the restriction error increases monotonically with stronger and stronger restrictions on X. Note that we must specify under which probability measure the expectations are taken.

Furthermore, we define the total error from making the two changes simultaneously as

$$a(X, \tilde{X}, P, \tilde{P}) := F(x_0^*(\tilde{X}, \tilde{P}), P) - F(x_0^*(X, P), P).$$

By adding and subtracting $F(x_0^*(\tilde{X}, P), P)$ in the right hand side above we see that the total error may be expressed as the sum of two errors:

$$a(X, \tilde{X}, P, \tilde{P}) = d_{\tilde{X}}(P, \tilde{P}) + r_P(X, \tilde{X}).$$

Note that we, in this expression, must use the discretization error measured using the restricted feasible set $\tilde{X}$. This means that we may actually get a smaller total error by restricting the feasible set from $\tilde{X}$ to $\bar{X}$, as long as an increase in the restriction error is compensated by a decrease in the discretization error

Note further that while the discretization error will increase monotonically with stronger and stronger restrictions of the feasible set, the discretization error has no no such monotonicity property. It is however reasonable to assume that the discretization error will decrease with stronger restrictions on the feasible set, as a smaller set reduces the possibility to exploit imperfections in the scenario tree. Hence the total error should (naïvely illustrated) look approximately like the curve in Figure 1.

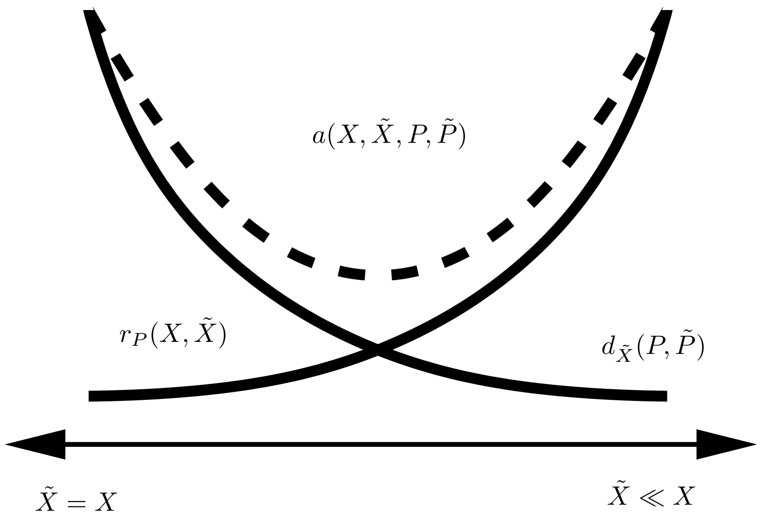
$a(X, \tilde{X}, P, \tilde{P})$
$r_P(X, \tilde{X})$
$d_{\tilde{X}}(P, \tilde{P})$
$\tilde{X} = X$
$\tilde{X} \ll X$

Figure 1: Changes in the restriction and discretization error.

3.3 Errors for multi-stage problems

When we move to three-stage problems, the situation becomes more complex. When a multi-stage stochastic programming model is used, it is used in a rolling fashion. One set of simplifications of the feasible set and probability measure is used to obtain the first-stage solution. Once the first random outcome is revealed, a new scenario tree is generated conditionally on this outcome, and the second stage decision is obtained, possibly using another restriction of the feasible set. The first-stage solution clearly is influenced by the choice of probability measure and the choice of feasible set for the first, second and third stages. The second stage solution is influenced by the first-stage solution, the probability measure chosen, and the feasible set for the second and third stages. Note that the second stage feasible set used to obtain the first-stage solution may be different from the second stage feasible set used to obtain the second stage solution. For multi-stage problems, the situation becomes even more complicated, and hence we abstain from deriving expressions for these errors, settling for a less rigorous approach.

If we look at the problem (1), we see that we may define the values of the recourse-problems recursively as

$$\begin{aligned} g_T(\overrightarrow{x}_{T-1}, \xi_T, X, P) &:= \min_{x_T \in X_T(\overrightarrow{x}_{T-1}, \xi_T)} f_T(\overrightarrow{x}_{T-1}, x_T, \xi_T) && \text{(5a)} \\ g_t(\overrightarrow{x}_{t-1}, \xi_t, X, P) &:= \min_{x_t \in X_t(\overrightarrow{x}_{t-1}, \xi_t)} (f_t(\overrightarrow{x}_{t-1}, x_t, \xi_t) + \\ &\quad \mathsf{E}^P_{\xi_{t+1}|\xi_t}[g_{t+1}(\overrightarrow{x}_{t-1}, x_t, \xi_{t+1}, X, P)]). && \text{(5b)} \end{aligned}$$

We use this recursion to define the objective function of the deterministic equivalent problem as

$$F_0(x_0, X, P) := f_0(x_0) + \mathsf{E}^P_{\xi_1}[g_1(x_0, \xi_1, X, P)]. \tag{6}$$

In the previous section all errors in F stemmed from the probability measure used; the function g was assumed to be correctly specified. In a multi-stage problem this can no longer be true, as the choice of probability measure and feasible set will affect g_1. When we use the function F_0 to obtain a first-stage solution, we should choose our reduced set, $\tilde{X}$, and probability measure, $\tilde{P}$, so that $g_1(x_0, \xi_1, \tilde{X}, \tilde{P})$ is a good approximation of $g_1(x_0, \xi_1, X, P)$, since errors in the definition of g_1 will induce errors in the first-stage solution. In the same fashion, errors in g_2 will induce errors in g_1. Hence, at stage t, the size of the feasible set should match the quality of the scenario tree (i.e., the number of branches) at this stage if we are to obtain as good an estimate of g_t as is possible. As it is common to use scenario trees with a higher number of branches near the root, this indicates that the feasible sets should be smaller and smaller further down the scenario tree, in order for the feasible sets to match the branching of the scenario tree.

3.4 Discretization

If the space of possible outcomes Ω in (1) above is infinite, then so is the number of possible decisions $x_T(\xi_T)$ that need to be considered when solving this problem. As mentioned above in Section 3 the space of all possible random outcomes must be discretized in order to make the problem computationally tractable. Many discretization methods exist, ranging from simple ones such as random sampling from the distribution (if the distribution is known) and random sampling augmented by variance reduction techniques (see Higle [12]), to complex schemes where the distance from the discretization $\tilde{P}$ to the probability measure P is minimized (see Pflug [19]). A good overview of different scenario tree generation techniques is given by Kaut and Wallace [16].

In this work we fit the scenario tree using optimization, as suggested by Høyland and Wallace [15], as Kouwenberg [17] has found this method to perform well without being overly complex.

3.5 Asset class selection

A fundamental part of the specification of an ALM model is the consideration of which assets or classes of assets to consider. A small number of assets will not give enough freedom to the model to find a good solution. The extreme case is a single asset, reducing the problem to simulating the development of a strategy consisting of periodically rebalancing the portfolio to a predetermined asset mix. Here, no optimization is done at all; the only feasible solution is the optimal solution. Towards the other end of the scale, adding a large number of assets will give the model more freedom, but requires us to make the scenario tree wider in order to capture the probability distributions of different investment outcomes accurately enough. A common requirement on a scenario tree is that it is free from arbitrage opportunities. In order to achieve freedom from arbitrage, the number of branches in the scenario tree at every point must be at least as many as the number of assets, accentuating the need for wider scenario trees when the number of assets increases.

Therefore, an important point of discussion when specifying our model is which assets or asset classes to choose. Traditionally, asset classes are aggregated according to category; one aggregate asset is created for stocks, one for bonds, one for real estate and so forth, possibly further divided by markets. This kind of division is used in the models described in [5, 4, 6], [13], [9], [18], and [7]. The rationale behind doing so is usually that it simplifies the problem statement when laws and regulations pose restrictions on the ownership of different asset classes. However, nothing stipulates that the optimal aggregate asset classes must consist of only one type of asset (e.g., stock or bond). Even if we wish to retain the rule of only one type of asset in an aggregate class, we still need to choose which sub-assets to aggregate, and in which proportions. We believe that gains may be made by taking into consideration not only the assets' internal correlation, but also the covariances of the aggregate classes and the correlation between aggregate classes and the liabilities. We return to this question in Sections 5.5 and 5.6.

4 A simplified ALM model

In order to test the ideas presented above, we construct a simple ALM problem. We imagine a fund manager required to cover a reserve by investing given funds in different asset classes. In addition, money flows in and out of the fund in a deterministic manner. If the funds are not sufficient to cover the reserve, a penalty is imposed. The penalty is progressive and has a number of levels (security factors). For instance, one level requires the funds to exceed 115% of the reserve, and a penalty is imposed proportionally whenever the total assets owned does not exceed this level. This allows us to specify the problem as follows:

Notation

$\hat{T}$	Time horizon.
$T = 0, 1, \ldots, \hat{T}$	Set of time-stages.
$t \in T$	Decision stage.
I	Set of asset classes.
Q	Set of penalty levels.

Variables

x_i^t	Amount of assets class i owned at time t.
y_i^{t+}	Amount of assets class i bought at time t.
y_i^{t-}	Amount of assets class i sold at time t.
z_q^t	Violation of penalty at level q at time t.
v^t	Total assets owned before trade at time t (no trading is performed at the last stage).

Parameters

p^t	Net payment inflow/outflow at time t.
γ_i	Transaction cost of asset i.
ρ_i^t	Price development of asset i from time $t-1$ to time t.
$\bar{x}_i$	Initial assets.
s_q^t	Penalty of level q at time t.
f_q	Security factor of level q.
R^t	Reserve requirement at time t.

The ALM model

$$\begin{aligned}
\max \quad & \mathsf{E}[v^T - \sum_{t \in T} \sum_{q \in Q} s_q^t z_q^t], & & \text{(7a)} \\
\text{s. t.} \quad & \sum_{i \in I} [y_i^{t-}(1-\gamma_i) - y_i^{t+}(1+\gamma_i)] = p^t, & t \in T \setminus \{\hat{T}\}, & \text{(7b)} \\
& y_i^{1+} - y_i^{1-} + \bar{x}_i = x_i^1, & i \in I, & \text{(7c)} \\
& y_i^{t+} - y_i^{t-} + \rho_i^t x_i^{t-1} = x_i^t, & i \in I, t \in 2 \ldots \hat{T}-1, & \text{(7d)} \\
& p^1 + \sum_{i \in I} \bar{x}_i = v^1, & & \text{(7e)} \\
& p^{t+1} + \sum_{i \in I} \rho_i^{t+1} x_i^t = v^{t+1}, & t \in 1 \ldots \hat{T}-1, & \text{(7f)} \\
& z_q^t + v^t \geq f_q R^t, & t \in T, q \in Q, & \text{(7g)} \\
& x, y, z \geq 0. & & \text{(7h)}
\end{aligned}$$

In this problem, the objective function measures the total wealth at the terminal period minus the penalties incurred in all periods. The constraint (7b) is a cash balance constraint, guaranteeing that the net proceedings of purchases and sales equal the external inflow/outflow of capital. The asset development is modeled by (7c)–(7d) and states that the amount owned at a certain time is the amount owned at the previous time times the price development, to which trade is added. The amount of all assets owned before trade is given by the equations (7e)–(7f).

Finally, the equation (7g) states that we must cover the reserves by a certain margin, or face a penalty. Note that the last stage of the problem is simply an evaluation; no trade is done at this stage. As the restrictions of the feasible set consists of adding constraints on the assets owned after trade, no restrictions are made to the final stage, just as assumed when we derived the errors in Section 3.

4.1 Aggregating asset classes by constraints

As we wish to explore the effect of using a limited number of synthetic assets in problem (7), we explicitly add constraints to restrict which assets may be owned, creating a second version of the model described above. In this model we add constraints for all but one of the assets. The rationale behind excluding one asset is that if we were to add constraints for all assets, we would get a set of linearly dependent constraints, making the problem infeasible if the constants c_{ik} do not sum to exactly 1 for all k. As all assets owned now will belong to a synthetic asset, the variables x_i^t are replaced by x_{ik}^t, giving the amount owned of basic asset i belonging to synthetic asset k at time t. The problem is further modified by adding the following notation:

K	The set of synthetic assets, in which we may invest.
$\hat{I}$	All basic assets except one. One asset is exempt from this set in order not to create linearly dependent constraints.
w_k^t	The total value held in synthetic asset k after trade at time t.
c_{ik}	The prescribed fraction of synthetic asset k which should be held in the basic asset i.

For a fixed value of c the model now becomes:

$$g(c) := \quad \max \quad \mathsf{E}[v^T - \sum_{t\in T, q\in Q} s_q^t z_q^t], \tag{8a}$$

$$\text{s. t.} \quad \sum_{i\in I} y_i^{t-}(1-\gamma_i) - y_i^{t-}(1+\gamma_i) = p^t, \qquad t \in T \setminus \{\hat{T}\}, \tag{8b}$$

$$y_i^{1+} - y_i^{1-} + \bar{x}_i = \sum_{k\in K} x_{ik}^1, \quad i \in I, \tag{8c}$$

$$y_i^{t+} - y_i^{t-} + \rho_i^t \sum_{k\in K} x_{ik}^{t-1} = \sum_{k\in K} x_{ik}^t, \quad i \in I, t \in 2\ldots\hat{T}-1, \tag{8d}$$

$$p^1 + \sum_{i\in I} \bar{x}_i = v^1, \qquad i \in I, \tag{8e}$$

$$
\begin{aligned}
p^{t+1} + \sum_{i\in I} \rho_i^t \sum_{k\in K} x_{ik}^t &= v^{t+1}, && i \in I, t \in T \setminus \{\hat{T}\}, && \text{(8f)}\\
z_q^t + v^t &\geq f_q R^t, && t \in T, q \in Q, && \text{(8g)}\\
w_k^t &= \sum_{i\in I} x_{ik}^t, && t \in T \setminus \{\hat{T}\}, k \in K, && \text{(8h)}\\
w_k^t c_{ik} &= x_{ik}^t, && t \in T \setminus \{\hat{T}\}, i \in \hat{I}, k \in K, && \text{(8i)}\\
x, y, z &\geq 0. && && \text{(8j)}
\end{aligned}
$$

Here, the constraint (8h) aggregates the total assets owned in each of the synthetic assets and the constraint (8i) makes sure that the prescribed asset fractions are kept. As remarked above, we do not prescribe a fraction for the first basic asset, as prescribing fractions for all basic assets would give a set of linearly dependent constraints. Instead the first asset will absorb the funds not assigned to other assets, and hence a fraction of $1 - \sum_{i\in\hat{I}} c_{ik}$ will be invested in the first basic asset in synthetic asset k.

Naturally, creating synthetic assets by adding extra constraints will make the problem larger and harder to solve. As an example, we may take a problem instance used later in our numerical experiments. In these, we have a 5-stage problem with 7 assets and 6400 scenarios. If we use no artificial assets, the size of problem (7) is 30,569 rows and 86,379 columns. If we add 3 artificial asset classes, the dimension of the problem increases to 67,886 rows and 116,588 columns. Note that the approach with adding constraints to define the linear combinations of assets is not entirely necessary. It would be perfectly possible to use problem (7) directly, substituting the assets I by a set of synthetic assets $\tilde{I}$, each synthetic asset having price development $\tilde{\rho}_i^t$. The constants c_{ik} would then be used to give the price development of the synthetic assets as

$$\tilde{\rho}_k^t := \sum_{i\in I} c_{ik} \rho_i^t$$

Further below, we show how the optimal value $g(c)$ [defined by (8)] might be differentiated with respect to c (under certain conditions). In the alternative formulation, we may differentiate $g(c)$ with respect to $\tilde{\rho}$ and use basic calculus to obtain the derivatives with respect to c. The reason why we do not use this simplified formulation is twofold. Firstly, transaction costs will not be captured correctly in the reduced model formulation, as we may sell basic asset i as part of selling synthetic asset $k \in K$ while buying the same basic asset as a part of selling synthetic asset $\bar{k} \in K$. Secondly, for implementation reasons, the dual variables for the constraints (7d) and (7f) are not directly available when the model is solved using nested Benders decomposition, as these constraints span more than one time period. Both these concerns might be addressed, the first by formulating only the first stage using explicit constraints defining the linear asset combinations, and the second by improving the solver.

4.2 Optimization of synthetic assets

The problem (8) contains a number of parameters c_{ik} which define the synthetic assets. In order to determine a good set of synthetic assets, we consider the optimal value of problem (8) to be a function of these parameters and formulate the problem to

$$\begin{aligned} \underset{c}{\text{maximize}} \quad & g(c), && && (9a)\\ \text{subject to} \quad & \sum_{i\in\hat{I}} c_{ik} \leq 1-\epsilon && k \in K, && (9b)\\ & c_{ik} \geq 0 && i \in \hat{I}, k \in K. && (9c) \end{aligned}$$

The ϵ perturbation is present to avoid numerical difficulties. If the fractions of c sum to something larger than 1, the problem (7) will become infeasible, and if they sum to exactly 1 the constraints of (7) will become linearly dependent. In order to avoid these situations, we set ϵ to something a couple of orders of magnitude larger than the machine precision. Perturbing the constraint (9) in this fashion is equivalent to setting a lower bound of ϵ on the fraction of the available means that is invested in the first asset, something that will have a minor impact on our results as long as ϵ is small.

Note that the values of c are considered to be parameters in (8) but variables in (9). In order to optimize (9) with a gradient descent method, we need to differentiate $g(c)$ with respect to c, that is, differentiate the optimal value of (8) with respect to changes to elements in the constraint matrix. As is noted in Dantzig and Thapa [8] in Section 7.6, a derivative of the optimal value for the problem

$$\begin{aligned} \text{minimize} \quad & z = d^t x, && (10a)\\ \text{subject to} \quad & Ax = b, && (10b)\\ & x \geq 0, && (10c) \end{aligned}$$

with respect to the matrix coefficient A_{ij} is given by

$$\frac{\partial z^*}{\partial A_{ki}} = -\pi_k^* x_i^*,$$

subject to certain non-degeneracy conditions. In this expression, x^* and π^* denotes the optimal primal and dual solution solution to (10), respectively.

In order to optimize (9) we use a gradient projection approach combined with an Armijo step-length rule (see Bertsekas [2]) and terminate when the norm of the projected gradient of $g(c)$ falls below a certain threshold. We obtain the value of the gradient from the expressions above. In order to avoid being caught in a local minimum, the optimization procedure is restarted from a number of random positions.

5 Numerical tests

5.1 Questions

As mentioned previously, we would like to explore whether aggregating assets in different ways has a significant impact on the performance of our model, when the obtained solutions are applied to out-of-sample scenarios. The questions we pose are:

- If we reduce the number of assets allowed for trading, how many synthetic assets are appropriate?
- When assets are aggregated into wider classes, should the aggregation be applied for all stages or should the first stage be excluded?
- If we reduce the number of assets by aggregation, should this reduction be made as a partitioning, or should a single asset be allowed to belong to several classes of aggregate assets?
- When we are aggregating assets into synthetic assets, do we benefit significantly from using the procedure described in Section 4.2 compared to simply using Markowitz's method?

In the second question we only consider exempting one stage from the requirement to invest in the linear combinations. The reason for using linear combinations is to prevent the stochastic programming solution from chasing spurious profits present only in the scenario tree, resulting in bad solutions when the full underlying random distribution is considered. In the scenario trees used in this work as well as in others (see for instance Dempster and Consigli [7], and Høyland and Wallace [14]), the first stage has a larger number of branches than the rest of the scenario tree. The first stage is given a greater number of branches as the decision made in this node will be used, while the only value of solutions in later nodes are the effect these solutions have on the first stage. This means that the SP solution should have a lower tendency to chase spurious profits in the first stage of the scenario tree, and hence that it may prove beneficial not to restrict that stage.

5.2 Testing environment

In order to answer our questions, we construct a micro-world, in which there exists only 7 asset classes, which are to be used to cover one reserve requirement. The correlations and expected values for these assets are given in Table 4 and 5 in Appendix A. The correlations are taken from real-world data of Swedish assets, whereas the means are adjusted, as the rather short (10 year, 1990–2000) data sample used had higher yields than experience indicates are reasonable. (For instance, Swedish stocks yielded an average of 19% per year during the period 1990–2000, in contrast to the long term value of 9.5% for the period 1918–1990 as given by Frennberg and Hansson in [10].)

For this study, we assume that the assets and the reserves have a jointly log-normal distribution. (This assumption serves the purpose to simplify the scenario generation process.)

Scenario tree generation In order to reduce the spurious differences between different methods caused by badly specified scenario trees, we optimize the trees to fit the statistical properties of the underlying distribution. This fitting is done using Matlab, giving us a large set of optimized set of outcomes with 40, 20, 16, 10 and 4 members. When the number of descendants is larger than or equal to 16, we fit the four lowest moments, as well as the covariances between the assets. When 10 descendants are used, the three lowest moments and the covariances are fitted. With only 4 descendant nodes, the two lowest moments are fitted. A tree is then constructed by randomly picking from these sets of outcomes, to generate a tree of the desired size. If we are to have more than 40 branches in one node, we combine a number of sets of optimized outcomes. We may simplify the scenario generation process in this fashion since we assumed that the distribution of the asset yields is stationary, and hence that this distribution does not change from node to node.

Rolling horizon simulations In order to test whether there is a significant difference between different asset aggregation methods, we conduct rolling horizon simulations. In rolling horizon simulations, the effect of actually using the optimal solution of our ALM problem is investigated, using simulated scenarios. In these simulations, we use a time horizon of 4 years. The process works by generating a number of test scenarios using random sampling from the distribution of the random parameters. For each of these scenarios, we apply the following procedure:

0: Set $t = 0$ and go to step 2.

1: Use the sample-path and the state of the company just after the decision at time $t-1$ to generate the state of the company just before a decision is made at time t.

2: Use the state of the world of the current sample-path at time t to generate a scenario tree. The length of the scenario tree is adapted to last to the end of the simulation.

3: Optimize over the tree, generating a decision for the company at time t. Use this decision to determine the state of the company after the decision is made at time t. Store information of the state of the company just after the decision.

4: If $t < T$ then set $t := t + 1$ and go to 1.

5: Use the stored states of the company to determine total penalties and the terminal value of the company, which are used to evaluate the success of the scenario.

The value of a scenario is defined by the total asset value at the end of the simulation, from which we deduct all penalties incurred during the way.

In step 2 above, we stated that the length of the scenario tree should be adapted to the remaining length of the scenario. In order to reduce the length of the scenario tree, we remove the last stage and increase the number of branches in the first stage, keeping the number of scenarios constant.

The methods are tested on 3000 scenarios, for 4 periods of time. The trees used have 6400 scenarios, and the four period tree branches 16, 10, 10, 4 with shorter trees generated as previously described. In order to eliminate random contributions to the differences in performance between methods, the scenarios used and the scenario trees used are identical in all cases.

Test scenario generation and statistical test When performing rolling horizon simulation, the results of a single method will vary greatly between different test scenarios. In order to compensate for this disturbance, we will compare different methods on a scenario to scenario basis. To further reduce the errors, we will use antithetic samples for our test scenarios, and hence we will compare different methods by comparing their difference in performance over a pair of antithetic scenarios. Since the difference between the different methods used is rather small, we employ a T-test to determine if the mean difference between two methods is statistically significant. Hence we first average our results over the antithetic pairs, and then apply a pairwise T-test to the resulting samples, in order to determine if the observed difference is statistically significant. The test statistica reported in all tables is the probability of getting a larger deviation from 0 than the one obtained in the test, given that the two methods give the same mean value, evaluated over the continuous random distribution.

5.3 Question 1: on the reduction of the number of assets

Earlier we have described why it might be beneficial to reduce the number of assets held in an asset liability problem. In this section we try to determine the optimal number of linear combinations of assets to hold. We do this by testing using 1, 2, 3, 4, and 7 different asset combinations. When we are to hold 7 different assets classes, we simply use the original ALM problem (7). For the cases of 1–4 assets, we use the model described in (8).

In the optimization, the function $g(\cdot)$ in (9) is defined as the optimal value of the problem (8) evaluated over a 4 period tree. The tree over which we evaluate branches by 40, 16, 16 and 10 in the consecutive stages, giving the tree a total of 102,000 scenarios. This tree is chosen larger than the trees used in the simulations, in order to lower the impact of the choice of this tree on the overall solution.

As the problem (9) is probably not convex, we can not guarantee that the solution found is optimal. In our experiments we do however obtain approximately the same asset mixes a majority of the times when starting from a number of different random starting points. The number of local minimums does however seem to increase with the number of asset combinations, as may be expected.

Number of assets	1	2	3	4
2	0.0006			
3	0.0005	0.33		
4	0.0104	0.26	0.034	
7	0.029	0.67	0.40	0.73

Table 1: Test statistica value for different combinations.

If we observe the synthetic assets obtained in this fashion, we get the results in Figure 2. In this figure the synthetic assets are ordered in descending order according to their use in the root node of the asset search problem.

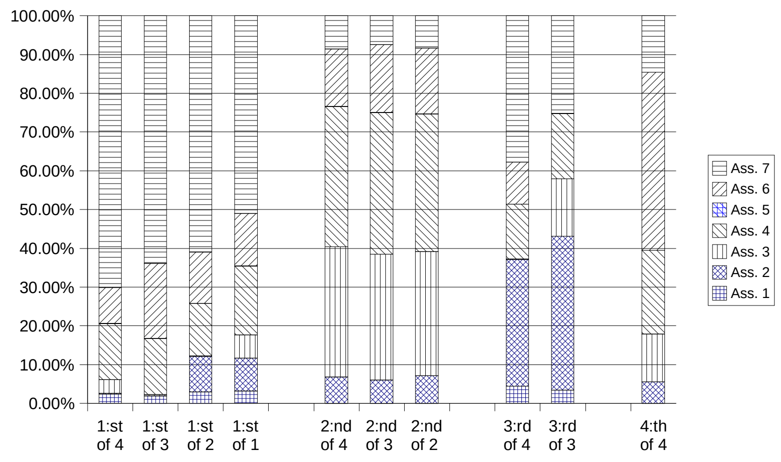

Figure 2: Funds used (ordered after decreasing use).

The funds obtained in the manner described above are then used unaltered throughout our simulations. In order to evaluate how well the chosen funds perform, we run 3000 simulations using the rolling horizon technique described in Section 5.2. The results from these runs are shown in Figure 3.

Although the number of different linear combinations is rather low, we see that we get the expected pattern of an increase in performance when the model is given a larger number of degrees of freedom, whereas the performance drops when the higher degree of freedom allows the solution to start adapting to the specific scenario tree used. As the difference between the different methods is rather small, we use the statistical test described above to generate Table 1. As might be seen from this table, not many of the differences are statistically significant when synthetic assets are applied at all stages (although the differences between using no synthetic assets and applying synthetic assets the second stage are significant).

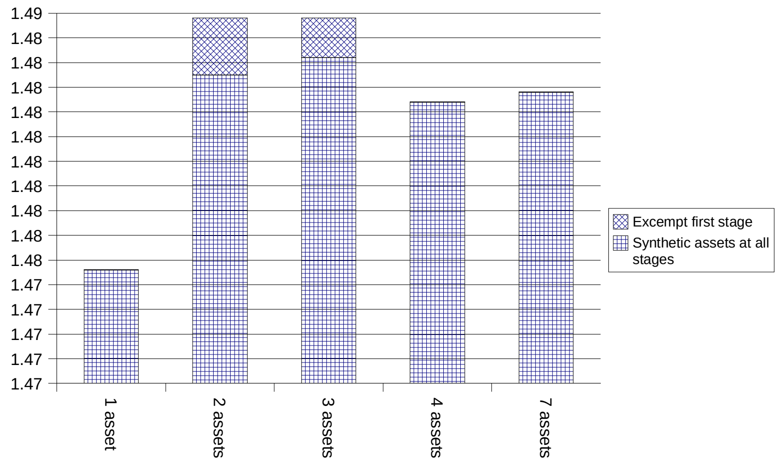

Figure 3: Results from asset reduction, simulation value as a function of the number of assets used.

Number of assets	gain	T-test statistica
2	0.0023	0.062
3	0.0016	0.17

Table 2: Gain from excluding stage 1 from synthetic assets.

5.4 Question 2: On when to apply asset combinations

As mentioned previously in Section 1.2 Gaivoronski and de Lange [11] experiment with an investment policy that is a hybrid between a fix-mix policy and a fully dynamic stochastic programming solution. They form this hybrid by letting the model freely choose an asset mix for the first two stages, but at the third and later stages, the asset mix i rebalanced to the asset mix chosen at the second stage. Inspired by this idea, we wish to see if performance is improved if we exempt the first stage from the requirement of owning only a limited number of linear combinations of assets. We try this for 2 and 3 linear combinations, as these number of combinations gave the best results in the previous case. The results are given in Figure 3 and Table 2.

As might be seen from Table 2 our results show that performance is gained from excepting the first stage from using synthetic assets. Well worth noting is that if we compare the use of two synthetic assets applied at stage 2 to no use of synthetic assets from the previous question, the difference in objective function value is 0.0061. Using the same pairwise test as earlier, the probability of getting a larger difference given that the two strategies are equivalent is 5e-4, showing that we most certainly do gain from using synthetic assets. In Figure 3 we now see the expected pattern of a top in performance for a somewhat restricted feasible set, in between a problem with a highly restricted feasible set, and a problem with no restrictions on the feasible set.

The difference in objective function value might seem small; it corresponds to increasing the yield of the assets by 0.1% without an increase in risk. Although this difference is small, it is not negligible.

5.5 Question 3: On strict partitioning versus overlapping synthetic assets.

As mentioned earlier, when aggregating assets into larger funds which are to be used at the strategic level of the asset allocation, it is not clear that the division into aggregate classes should be made as a partitioning, with each basic asset belonging to only one synthetic asset. On the contrary, when the synthetic assets were optimized in the previous case, for the case of two funds, both funds contained more than 5% of assets 2, 4, 6 and 7. In order to test whether forcing the synthetic assets to define partitions will make the solution better or worse, we divide the assets into high-risk (assets 3,4 and 6) and low risk (assets 1, 2, 5 and 7), based on the standard deviation of their yields. We now optimize two synthetic assets in the same fashion as in case 1, while making sure that one fund contains only high-risk assets, and one fund contains only low-risk assets by imposing constraints on c. The funds so obtained are given in Figure 4. For comparison we show the synthetic assets obtained when a partitioning is not required, and the synthetic assets are allowed to overlap.

As the previous case indicated that excepting the first stage from using synthetic assets was beneficial, we do so in this case as well. When we do not allow the synthetic assets to overlap, the difference in performance between the two divisions becomes $1.0e{-}4$ in favor of the partitioning. The T-test gives us that the probability of obtaining a larger difference if the two methods are equivalent is 0.94. Hence there does not seem to be any difference if we allow the assets to overlap or not.

5.6 Question 4: On the usefulness of optimizing aggregate assets within a scenario tree

The previous question concerned the effects of aggregating assets as a strict partitioning. The proportions of the basic assets to be included in the synthetic assets were determined via optimization, where we tried to find the synthetic assets which gave the best results when used in problem (8). As this procedure is rather complex, we would like to determine if we might get better, or at least not significantly worse results, by using the classical Markowitz optimization procedure. In order to compare asset classes created using optimization over the tree and asset classes created using the Markowitz method, we naturally need to construct asset classes using the latter method. In order to make a comparison with optimization over the

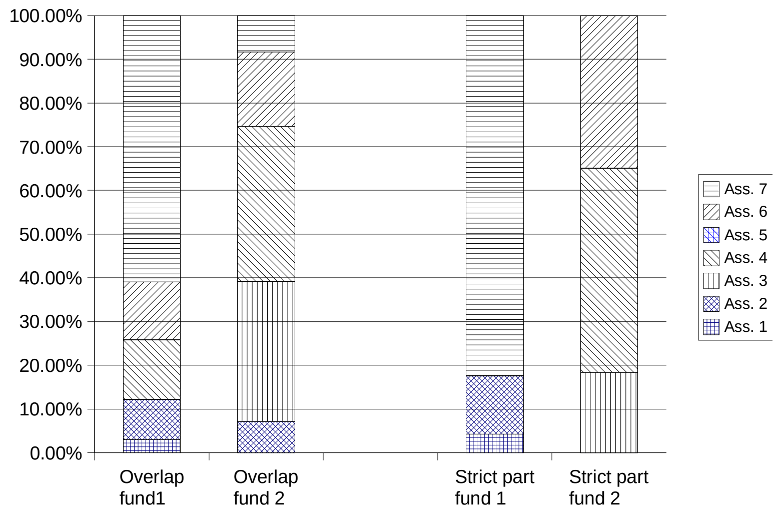

Figure 4: Funds used (partitioning versus with overlap).

tree possible, we make sure that the low-yield and the high yield assets constructed via Markowitz optimization has the same expected yield as the ones generated using strict partitioning in Section 5.5 above. The Markowitz optimization is performed in two ways. In the first case, we assume that we have one unit of money to invest in the assets, and we do this in the way that will minimize the standard deviation of our yield, given that we get the desired average yield, an approach that will ignore the reserve requirements. In the second case, we assume that we have one unit of assets which is used to cover one unit of the reserve, and we try to minimize the standard deviation of the surplus. These two methods may be seen as using the liability hedging credit of Sharpe and Tint [20], with the weight of the liability hedging credit set to 0 and 1, respectively.

Using the two versions of Markowitz optimization, we obtain the synthetic assets given in Figure 5. As previously, we run rolling horizon simulations for the two new sets of synthetic assets, and the results from these tests are given in Table 3. We see that there is no significant difference between using the Markowitz optimization with consideration to the reserve, and optimizing over the tree. There do however seem to be a difference between optimizing over the tree and using the Markowitz method without considering the reserve.

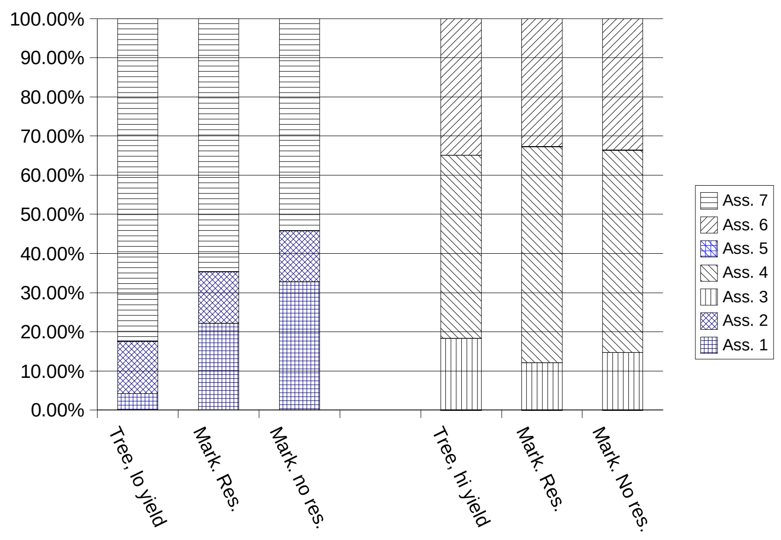

Figure 5: Funds found with Markowitz optimization, vs. funds from Section. 5.5.

Mean value		
Markowitz, reserve	Opt. over tree	Markowitz, no reserve
1.4847	1.4848	1.4827
Statistical significance		
	Opt. over tree	Markowitz, no reserve
Markowitz, no reserve	0.08	
Markowitz, reserve	0.89	1e-4

Table 3: Test statistica value for Markowitz optimization vs. optimization over tree.

6 Conclusions and further work

6.1 conclusions

In this article we have shown that reducing the number of asset classes by allowing trade in only a limited number of linear combinations of assets (termed synthetic assets) increases the performance of an asset liability model. We further found that the model should not be constrained to using the synthetic assets at the root node of the problem, as the higher number of branches commonly used at this stage should provide a good enough description of the possible random outcomes to make artificial restrictions of the feasible set superfluous, if not directly damaging.

Furthermore, the optimal number of assets was low; the best results where obtained using only two synthetic assets. In retrospect, this is not surprising. The main purpose of a multi-stage ALM model is to make it possible to change the risk-reward tradeoff in the future, depending on the state of the company and the world, and consider these future changes when todays decision is made. The simplest model allowing such changes is a model with two synthetic assets.

In addition to searching for the optimal number of synthetic assets, we examined if the aggregation into synthetic assets should be done as a partitioning, or if an overlap should be allowed. As we were not able to show any difference between no overlap and overlap, we conclude that this decision should be based on other criteria. (As it is simpler to use non-overlapping assets, we will probably do this in the future.)

The primary method of aggregating assets into synthetic assets in this work has been to find the synthetic assets which give the best objective value over an instance of the stochastic linear programming model. Our experiments do not indicate that this method outperforms the Markowitz portfolio optimization model, provided however that the latter is extended to take correlations with the reserve into account. From the experiments it is clear that there is a significant disadvantage to not include the reserve considerations when the synthetic assets are determined. If we in the future use the Markowitz model, we must however determine how the trade-off between risk and yield should be made when the synthetic assets are created. Right now the yield for the two synthetic assets were taken from the funds obtained via optimization over a scenario tree.

Well worth noting is that the increased performance we experienced when using synthetic assets is not the only benefit. As for most optimization problems, the computational prize of solving a stochastic programming problem increases with the dimension of the problem. If the number of assets in the problem is reduced, so is the size of the optimization problem. Hence aggregating assets makes it possible to increase the number of discretization points used in the scenario tree, or the number of stages in the tree, without increasing the size of the problem. As a better description will give more accurate solutions, aggregating asset classes would have a positive effect even if a solution from an aggregated problem performs just as well as an unaggregated one, since aggregating assets would allow us to solve problems with larger scenario trees without using more computational resources.

6.2 Further work

In this work, the asset fractions are enforced using artificial constraints, for reasons explained in Section 4.1. A better approach would probably be to let the model trade directly in the synthetic assets, as this significantly would reduce the number of variables.

Furthermore, as we did not find a significant advantage of using asset classes created by optimization over the scenario tree compared to using Markowitz portfolio selection method, it would probably be better to use the latter method, as it is simpler. However, it still leaves us with the problem of how to choose the value of risk used in this method. A possible approach is to use dual information from the stochastic programming ALM problem. In order to illustrate this approach, we assume that we have a simple two stage stochastic programming ALM problem with N scenarios. The problem includes only two synthetic assets and one reserve requirement. The yield of the assets for scenario i are given by the constants a_i and b_i, and the reserve for scenario i is given by s_i. In the way described in Section 4.2, we may obtain the change in optimal objective value if we make a marginal change to the parameter a_i; we denote this change a_i^*. By summing a_i^* over i we obtain the value of marginally changing the yield of the low-yield synthetic asset. In the same fashion, the expression

$$\frac{\sum_{i=1}^{N} a_i^*(a_i - \mathsf{E}[a])}{2}$$

gives the change in the objective function value if the variance of the low yield asset is increased marginally. Similarly, we may obtain the value of marginally changing the correlation between the low-yield asset and the reserve requirement, as well as information regarding the high-yield asset. This dual data may now be used in a Markowitz optimization to find new low yield and high yield assets, which are to be used in the stochastic programming ALM problem. As the value of correlating the yield to the reserve probably would be different for the low-yield and the high-yield asset, this is equivalent to using different liability hedging credits for the two synthetic assets.

In this work we have assumed that the random distributions are stationary. If this is not the case, we may have one Markowitz problem for each node in the scenario tree, adapting the low and high yield asset to the conditions in each node, possibly obtaining better performance.

Furthermore, in this work, we have only studied how the synthetic assets should be constructed in order to prevent the solution from chasing spurious profits. However, once the synthetic assets are determined, the scenario tree need no longer provide an accurate approximation of the correlations and yields of the basic assets, all that is needed is a scenario tree which accurately describes the moments and correlations of the created synthetic assets. Hence the scenario tree may be regenerated in order to provide better information on the synthetic asset, while ignoring statistical properties regarding the basic assets which are now irrelevant.

7 Acknowledgments

This work was partially funded by Nordea life & pension.

References

[1] F. ALTENSTEDT, *An asset liability management system for a Swedish life insurance company*, Preprint 2003:47, Chalmers University of Technology, Department of mathematics, SE-412 96 Göteborg, Sweden, 2003. Submitted to Annals of Operations Research.

[2] D. P. BERTSEKAS, *On the Goldstein–Levitin–Polyak gradient projection method*, IEEE Transactions on Automatic Control, AC-21 (1976), pp. 174–184.

[3] S. P. BRADLEY AND D. B. CRANE, *A dynamic model for bond portfolio management*, Management Science, 19 (1972), pp. 139–151.

[4] D. R. CARIÑO, D. H. MYERS, AND W. T. ZIEMBA, *Concepts, technical issues, and uses of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 450–462.

[5] D. R. CARIÑO AND W. T. ZIEMBA, *Formulation of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 433–449.

[6] D. R. CARIÑO, W. T. ZIEMBA, T. KENT, D. H. MYERS, C. STACEY, M. SYLVANUS, A. L. TURNER, AND K. WATANABE, *The Russell–Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming*, Interfaces, 24 (1994), pp. 29–49.

[7] G. CONSIGLI AND M. DEMPSTER, *Dynamic stochastic programming for asset–liability management*, Annals of Operations Research, 81 (1998), pp. 131–161.

[8] G. B. DANTZIG AND M. N. THAPA, *Linear Programming*, vol. 1, Springer-Verlag, New York, 1997.

[9] C. DERT, *Asset liability management for pension funds; A multistage chance constrained programming approach*, PhD thesis, Erasmus University Rotterdam, 1995.

[10] P. FRENNBERG AND B. HANSSON, *Swedish stocks, bonds, bills and inflation*, Applied Financial Economics, 2 (1992), pp. 79–86.

[11] A. A. GAIVORONSKI AND P. E. DE LANGE, *An asset liability management model for casualty insurers: complexity reduction vs. parameterized decision rules*, Annals of Operations Research, 99 (2000), pp. 227–250.

[12] J. L. HIGLE, *Variance reduction and objective function evaluation in stochastic linear programs*, INFORMS Journal on Computing, 10 (1998), pp. 236–247.

[13] K. HØYLAND, *Asset liability management for a life insurance company: A stochastic programming approach*, PhD thesis, Department of Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway, 1998.

[14] K. HØYLAND AND S. W. WALLACE, *Analyzing legal regulations in the Norwegian life insurance business using a multistage asset-liability management model*, European Journal of Operational Research, 134 (2001), pp. 293–308.

[15] ——, *Generating scenario trees for multistage decision problems*, Management Science, 47 (2001), pp. 295–307.

[16] M. Kaut and S. W. Wallace, *Evaluation of scenario-generation methods for stochastic programming.* Available from `http://www.iot.ntnu.no/~mkaut/CV_and_study/SG_evaluation.pdf`.

[17] R. Kouwenberg, *Scenario generation and stochastic programming models for asset liability management*, European Journal of Operational Research, 134 (2001), pp. 279–292.

[18] J. M. Mulvey and H. Vladimirou, *Stochastic network programming for financial planning problems*, Mangement Science, 38 (1992), pp. 1642–1664.

[19] G. Pflug, *Scenario tree generation for multiperiod financial optimization by optimal discretisation*, Mathematical Programming, Series B, 89 (2001), pp. 251–271.

[20] W. F. Sharpe and L. G. Tint, *Liabilities-a new approach*, the journal of portfolio management, (1990), pp. 5–10.

[21] K. J. Worzel, C. Vassiadou-Zeniou, and S. A. Zenios, *Integrated simulation and optimization models for tracking indices of fixed-income securities*, Operations Research, 42 (1994), pp. 223–233.

A Simulation Data

A.1 Simulation parameters

$\bar{x}_i$	$\{0.1, 0.2, 0.2, 0.1, 0.1, 0.1, 0.1\}$
γ_i	$0.005, i \in I$
p^t	$0.06, t \in 0, \ldots, \hat{T}$
f_q	$\{1.15, 1.06, 1.02, 1.00\}$
s_q	$\{1.0, 1.0, 2.0, 2.0\}$

A.2 Assets

The assets used in the simulations have the following means and standard deviations:

	reserve	a1	a2	a3	a4	a5	a6	a7
mean	11.01	6.21	7.38	12.48	11.37	4.59	8.19	6.18
Std.	1.88	5.26	9.46	24.81	18.09	0.43	16.06	3.52

Table 4: Mean and standard deviation of assets used (%).

	reserve	a1	a2	a3	a4	a5	a6
a1	0.49381						
a2	0.18627	0.18943					
a3	0.45105	0.45642	0.20575				
a4	0.33364	0.33703	0.58166	0.67816			
a5	0.56793	0.61823	0.11801	0.26933	0.19448		
a6	0.48187	0.48950	0.03016	0.15257	0.01430	0.29368	
a7	0.19084	0.19593	0.17999	0.21120	0.09766	0.16223	0.17333

Table 5: Correlations of assets used.

Paper IV

Policy optimization: Parameterized Decision Rules vs. Stochastic Programming for Asset Liability Management

Fredrik Altenstedt and Michael Patriksson
Department of Mathemathics
Chalmers University of Technology
412 96 Göteborg, Sweden

September 19, 2003

Abstract

Stochastic linear programming and the optimization of parameterized policies are two techniques which may be used for asset liability management (ALM). In both cases the randomness of the problem is addressed by generating a scenario tree and optimizing the ALM problem over this tree. The principal difference between the methods is the degree of coupling between different scenarios in the tree. In stochastic linear programming there exists little or no such coupling, making the resulting problem relatively simple to solve; however stochastic linear programming requires scenario trees of high quality to perform well. Parameterized policies have a stronger coupling between the scenarios, making the problem harder to solve; this stronger coupling however makes it possible to use smaller scenario trees. We construct a simple ALM problem in order to compare stochastic programming to parameterized policies for a problem with many time-periods as previous comparisons in the literature have had few enough time-stages to make the conditions ideally suited for stochastic programming. We further show that by combining the two approaches into a hybrid approach, a superior performance may be achieved, as the respective strengths of the two methods are utilized to complement each other.

Keywords: Stochastic programming; Asset liability management; Optimization; Parameterized policies

1 Introduction

1.1 Background

As the name implies, asset liability management (ALM) is the art of jointly managing assets and liabilities. The driving idea is that by treating assets and liabilities in an integrated fashion decisions leading to lower total risk may be taken. Stochastic programming (SP), both linear and non-linear, has become an important tool for ALM. A number of models are described by, amongst others, Cariño et al. [7, 6, 8], who treat the problem of optimally allocating investments for a Japanese insurance company, Consigli and Dempster [10], who have formulated a stochastic linear programming model to be used by managers of an insurance fund, Høyland and Wallace [17, 18],who have treated the problem of managing assets for a Norwegian life insurance company, and Dert [11] and Kouwenberg [21], who treat Dutch pension funds. The common denominator in these articles is that they all address the problem of a fund or company (from now on we assume that it is a company) investing means to cover future liabilities.

An important feature of ALM models implemented by stochastic programming is that the models are dynamic: when making investment decisions a stochastic programming model explicitly takes into account that the decisions may, and probably will, change as time goes by. This dynamic characteristic of stochastic programming models makes it possible for them to outperform static methods such as the well known Markowitz portfolio method. (When funds are optimized using the Markowitz method, the asset mix is assumed to be constant during the time-span considered, which is generally not optimal.) The downside of this dynamic property is that stochastic programming is complex and computationally demanding.

Another, simpler way of modeling dynamic decisions is the use of parameterized decision rules. A decision rule is a multivariate function mapping each possible state of the world and the managed company to a decision to take. As the combined state of the company, state of the world and the decision to take, may be expressed as vectors of real numbers, the decision rule is nothing else than a vector-valued function from the space of possible states to the space of decisions. If the rule is parameterized, then the input to the function consists of both the state and a number of parameters. Parameterized policies are interesting since we may tune the parameters to obtain good decisions for a given set of inputs. A number of well known simple strategies, such as the buy-and-hold and fixed-mix strategies, are actually parameterized decision rules, albeit simple ones. Those strategies are however not dynamic since they give the same decision regardless of the state of the company, and hence they do not account for the company's own future decisions.

An example of a dynamic decision rule taking the state of the company explicitly into account is the constant proportional portfolio insurance rule (CPPI), described by Perold and Sharpe [24]. This rule can be used when there is a bound below which the total value of our investments should not fall, and we may invest in one risk-less asset and one risky asset. The rule states that if we own v and do not want the total assets to fall below f, we should invest a fraction of $\max\{0, d \cdot (v - f)\}$ into the risky asset (d is a proportionality constant, and hence the policy has two parameters, d and f). This rule directly considers the state of the company, as the difference between the current asset value and the floor level is the basis for our decisions. Systems where parameterized decision rules are used are described by Mulvey, Gould and Morgan [23], as well as Boender [5], although neither of the articles gives much information on how the policies are formulated or optimized.

1.2 Motivation and overview

Our interest in parameterized decision rules originates in our work on an asset liability management system for a Swedish life insurance company [2]. In order to address its problem, we have constructed an ALM model based on multi-stage stochastic programming. Our experiences with this model have however lead us to identify some drawbacks with the SP approach. Like several researchers before us, we found that the optimal solution to a stochastic linear programming model fluctuates significantly with the scenario tree chosen to represent the random variables.

As a remedy to such fluctuations, several methods have been devised to make the scenario tree a better representation of the underlying true random distribution. Some of these techniques are: variance reduction (described by Higle [15]), an aggregation of a large number of scenarios while retaining the basic stochastic properties (used by Klaassen [20] and Cariño, Myer and Ziemba [6]), an optimization of the scenario tree to make it fit the desired statistical properties (used by Høyland and Wallace [19] as well as Kouwenberg [21]) and an optimization of the scenario tree to minimize the distance between the tree and the underlying distribution (used by Pflug [25]). These methods do indeed improve the situation, although a significant difference between the solutions obtained using different scenario trees still exist, even when they are constructed to exactly mimic the covariances as well as the first four central moments of the underlying distribution. Naturally, the problem of fluctuating solutions may be reduced by increasing the number of scenarios, but this will make the problems harder to solve.

Our partner company wishes to use a higher number of assets in their model than what is currently common in the literature. This wish entails a need to widthen the scenario trees in order to correctly describe the covariances and moments of the underlying distribution, which in turn means that the size of the scenario trees makes the model impractical to use, at least for testing purposes. In a previous work we have studied methods for aggregating different assets into larger synthetic assets in order to reduce the dimension of the problem (see [1]). This technique may be used to allow a larger set of assets without increasing the size of the scenario tree.

However, asset aggregation does not help us to solve another serious problem of a stochastic programming based approach, namely how to interpret the results. Stochastic programming generally works as a closed black box. The user specifies his/her risk preference and subjective view of the random distribution of the uncertain parameters, and if the model is specified correctly, it will provide the user with a decision which is consistent with his/her beliefs. A parameterized decision rule is easier to interpret, since the entire rule is explicitly available in the form of the aforementioned policy function.

Parameterized rules do however have their own drawbacks. Firstly, while the parameters may be optimized using suitable methods, we still face the problem to choose the shape of the parameterized policy. (Should it be a piecewise linear function, a polynomial of the state variables, etc.?) As this problem involves choosing between different types of functions, it is not suited for automatic optimization. Secondly, it becomes almost impossible to find an optimal or near optimal set of parameters if the number of parameters becomes too large, or if the policy becomes too complex.

In order to try to mitigate the respective weaknesses of the two methods, we attempt a hybrid approach. This approach consists of constructing a policy function from a number of stochastic programming solutions. A stochastic programming problem is optimized for a number of different states of the world; the optimal solutions together with the state under which they are obtained are collected in a database. This collection of solutions are then used to create a policy function in the following fashion: for a given state of the world and the company, we find a set of states in the database which are close to this given state. The optimal solutions corresponding to the chosen states are then interpolated to form the output of the policy function. Note that this requires the definition of a metric on the space of possible states. In this work, the metric is very simple, the distance between two states is given by the consolidation (assets over liabilities).

When we use stochastic programming to address an ALM problem, the model must be simplified in order to become computationally tractable. Among other things we typically need to reduce the number of time-stages in the problem, either by removing the later stages or by aggregating time-stages. This kind of simplifications will most certainly introduce some sort of bias into the solution, towards, for example, too risky or too conservative investments. In previous comparisons made by Kouwenberg [21], and Fleten, Høyland and Wallace [12], the length of the test scenarios have been short enough to make it feasible to have one decision stage in the stochastic programming model for each decision stage in the test scenario. In the comparisons made by Golub et al. [13], a two-stage model is used but decisions are made for each time-stage. This is equivalent to using a multi-stage tree which only branches at the root node.

In order to decrease the bias caused by stage aggregation, we construct a parameterized policy from the table-based policy by adding to it a low-dimensional which is controlled by a set of parameters. Similar to a pure parameterized policy, the parameters are then optimized over a large number of scenarios in order to find an optimal disturbance.

By combining stochastic linear programming and parameterized policies in this fashion, we hope to utilize the advantages of both methods to our benefit. Stochastic programming is used to produce the general shape of a parameterized policy, which reduces the problem of choosing a shape for our policy function to that of choosing a shape of the disturbance. As the policy is computationally cheap to evaluate, it is possible to test the solution using a large number of scenarios, which is not possible for a pure stochastic programming approach. Furthermore, since the policy is explicitly available, albeit in the form of a set of tables, it is easier to evaluate and use than a system based on pure linear stochastic programming. To our knowledge, this is the first time stochastic linear programming and parameterized policies are combined.

1.3 Outline

In order to address an ALM problem using both stochastic linear programming and parameterized policies, we start by describing a general ALM problem in Section 2. In Section 3 we describe the two methods we will compare and in Section 4 we make a qualitative comparison between them. The simple ALM problem used for tests as well as the tests themselves are described in Section 5, whereas conclusions are drawn from these tests in Section 6.

2 Formulation of an ALM problem

2.1 A mathematical model

As we previously mentioned, ALM problems are solved to make optimal investments in assets in order to cover future liabilities. In reality reinvestment decisions are made continuously in time (although markets may close, etcetera) but in order to simplify the problem we assume that trade is only possible at a finite number of points in time. Under this assumption, we may formulate the general ALM problem as follows:

$$\begin{aligned}
\min_{y_0} \quad & f_0(x_0) + \mathsf{E}_{\xi_1} \min_{y_1(\xi_1)} [f_1(\overrightarrow{x}_1(\xi_1), \xi_1) + \mathsf{E}_{\xi_2|\xi_1} [\min_{y_2(\xi_2)} f_2(\overrightarrow{x}_2(\xi_2), \xi_2) \\
& + \cdots + \mathsf{E}_{\xi_T|\xi_{T-1}} [\min_{y_T(\xi_T)} f_T(\overrightarrow{x}_T(\xi_T), \xi_T)] \cdots]], && \text{(1a)} \\
\text{s. t.} \quad & x_0 = g(\bar{x}, y_0), && \text{(1b)} \\
& y_0 \in Y_0, && \text{(1c)} \\
& x_t(\xi_t) = g(x_{t-1}(\xi_{t-1}), y_t(\xi_t), \xi_t), \qquad t = 1, \ldots, T, && \text{(1d)} \\
& y_t(\xi_t) \in Y_t(x_{t-1}(\xi_{t-1}), \xi_t), \qquad t = 1, \ldots, T, && \text{(1e)}
\end{aligned}$$

where all constraints hold almost surely, and where the variables and parameters are defined as:

T	last time-stage,
ξ_t	all the random information known at time t,
$x_t(\xi_t)$	the state of the company at time t,
$\overrightarrow{x}_t(\xi_t)$	the all states of the company up to time t,
$\bar{x}$	the initial state of the company,
$y_t(\xi_t)$	the decision made at time t,
$Y_t(x_{t-1}(\xi_{t-1}), \xi_t)$	the feasible set at time t, which is dependent on random information and the previous state of the company.
$g(x_{t-1}(\xi_{t-1}), y_t(\xi_t), \xi_t)$	the transition function, giving the state at time t as a function of the state at time $t-1$, the random outcome at time t, and the decision made at time t.

In this model, we assume that the decision y_t is taken after the random outcomes at time t become known; hence, y_t may depend on these outcomes. Furthermore, we assume that the variables of the model may be divided into two categories: State variables (x_t) and decision (or control) variables (y_t), and that the state variables are explicitly determined by the control variables.

2.2 Scenario trees and discretization

In order to make the problem (1) computationally tractable, we need to restrict the space of possible outcomes of the process $\{\xi_t\}$ to a finite set of manageable size. This is performed by discretizing the possible outcomes to a scenario tree, using one of the methods mentioned in Section 1.2. The root node of this tree will contain the certain information available at time 0 (the initial state of the company, x_0, and the known point of origin of the process, ξ_0). Since the process has been discretized, there exists a finite number of possible outcomes of ξ_1, each forming a node in the tree, having the root node as its parent. In the same fashion, each outcome node i at stage $t < T$ will have a set $C(i)$ of successor outcome nodes at time $t + 1$. Furthermore, we collect all the nodes of the tree into the set N, all the nodes at time t into the set N_t, and let the operator $p(i) : N \setminus N_0 \to N \setminus N_T$ denote the parent of an outcome. The unconditional probability for the outcome in node i to occur is defined by P^i.

3 Two specifications of the ALM problem

3.1 Stochastic linear programming

In stochastic linear programming, all constraints of the program (1) are assumed to be linear, and the sets Y_t are assumed to be polytopes. If we further assume that the distribution of the possible random outcomes is finite (discretized according to the previous section) then we may write the deterministic equivalent linear programming problem as that to

$$\begin{aligned}
\min_{y^i, i\in N} \quad & \sum_{i\in N} P^i f^i(x^i), && && \text{(2a)}\\
\text{s. t.} \quad & x^0 = F^0 y^0 + G^0 \bar{x} + e^0, && && \text{(2b)}\\
& B^i y^i \le b^i, && i \in N, && \text{(2c)}\\
& x^i = F^i y^i + G^i x^{p(i)} + e^i, && i \in N \setminus N_0, && \text{(2d)}
\end{aligned}$$

where the following notation is introduced:

x^i	the state of the company after decisions are made in node i,
y^i	the decision made in node i,
d^i, A^i, B^i, F^i, G^i	the random outcome in node i,

$f^i(\cdot)$	a piecewise linear convex objective function in node i.

As may be seen from the formulation this is a large scale linear programming problem, which may be solved directly or by using decomposition methods, such as the Benders and nested Benders decomposition methods [4, 26].

3.2 Parameterized policies

Applying parameterized policies to the problem (1) yields the problem to

$$
\begin{aligned}
\min_{\alpha} \quad & f_0(x_0) + \mathsf{E}_{\xi_1}[f_1(\overrightarrow{x}_1(\xi_1), \xi_1) + \mathsf{E}_{\xi_2|\xi_1}[f_2(\overrightarrow{x}_2(\xi_2), \xi_2) \\
& + \cdots + \mathsf{E}_{\xi_T|\xi_{T-1}}[f_T(\overrightarrow{x}_T(\xi_T), \xi_T)] \cdots]], && && \text{(3a)} \\
\text{s. t.} \quad & x_0 = g(\bar{x}, y_0), && && \text{(3b)} \\
& y_0 \in Y_0, && && \text{(3c)} \\
& x_t(\xi_t) = g(x_{t-1}(\xi_{t-1}), y_t(\xi_t), \xi_t), && t = 1, \ldots, T, && \text{(3d)} \\
& y_t(\xi_t) \in Y_t(x_{t-1}(\xi_{t-1}), \xi_t), && t = 1, \ldots, T, && \text{(3e)} \\
& y_t(\xi_t) = \gamma(x_{t-1}(\xi_{t-1}), \xi_t, t, \alpha), && t = 1, \ldots, T, && \text{(3f)}
\end{aligned}
$$

including the following additional notation:

α	a vector of policy parameters,
$\gamma(x_{t-1}(\xi_{t-1}), \xi_t, t, \alpha)$	the policy function.

This problem extends the problem (1) with the addition of the constraint (3f) in which we lock the value of y_t to the value of the policy function γ. Choosing the function γ reduces the degrees of freedom of the problem from the dimension of y times the number of possible outcomes to the dimension of the parameter vector α. Similarly, we may apply parameterized policies to the problem (2) by adding the constraint (3f). [Comparing the two problems (2) and (3), the constraint (2b) corresponds to (3b), (2c) corresponds to (3c) and (3d), and (2d) corresponds to (3e).]

4 A qualitative comparison

In this section we describe the major differences between stochastic linear programming and parameterized policies, as well as give a qualitative description of the strengths and weaknesses of the two methods.

4.1 Local vs. global information

Looking at the problem (2), we see that the state of the company (x) at a node is dependent of the state of the company at the parent node. Since the state x at a node is also influenced by the random information at that node, this means that the random information at a node will directly influence the states of the descendant subtree. In the same fashion, the state at a node depends on the decision y taken at

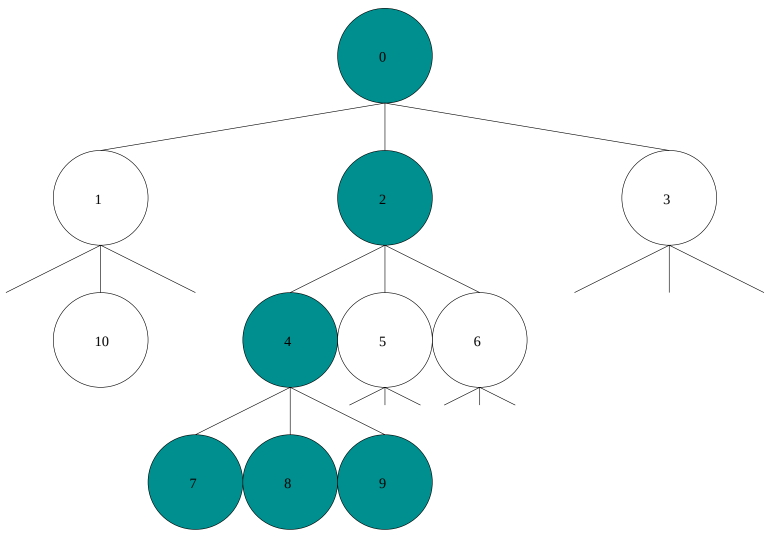

Figure 1: Scenario tree marking random information affecting node 4, stochastic linear programming.

a node. Hence, an optimal decision at a node i is directly influenced by the random information at the ancestor nodes directly above node i in the tree, as well as the random information in the descendant subtree. This is illustrated in Figure 1, where the nodes influencing node 4 are shaded. To get a reasonably good solution at this node, its children must give a sufficiently accurate representation of the space of all possible outcomes of the random parameters, conditioned on the state at this node. For example, in Figure 1 it is perfectly possible that the decisions at nodes 0, 1 and 2 imply that the company has identical states at nodes 10 and 4. In the same fashion it is perfectly possible that the state of the world is identical (or, nearly the same) at both nodes 10 and 4. Given that the states of both the company and the state of the world at these nodes are similar, the decision made at these nodes should be similar. In a stochastic programming formulation, however, there is no guarantee that the solutions at these nodes will be identical, or even close, as their sets of child outcomes may differ. We conclude that a stochastic programming solution directly uses only a small part of the scenario tree when determining the solution for a single node.

Looking at the parameterized policy approach for the same example shows that there is a direct coupling between the two nodes 10 and 4 via the policy function γ, illustrated in Figure 2. As this function considers only the states of the world and the company at a node, the decisions made at two nodes with identical states will be identical. In addition, if the policy function is continuous, it will yield similar decisions for nodes with similar states. Hence, a parameterized policy approach utilizes information from the whole tree when determining the solution for a single

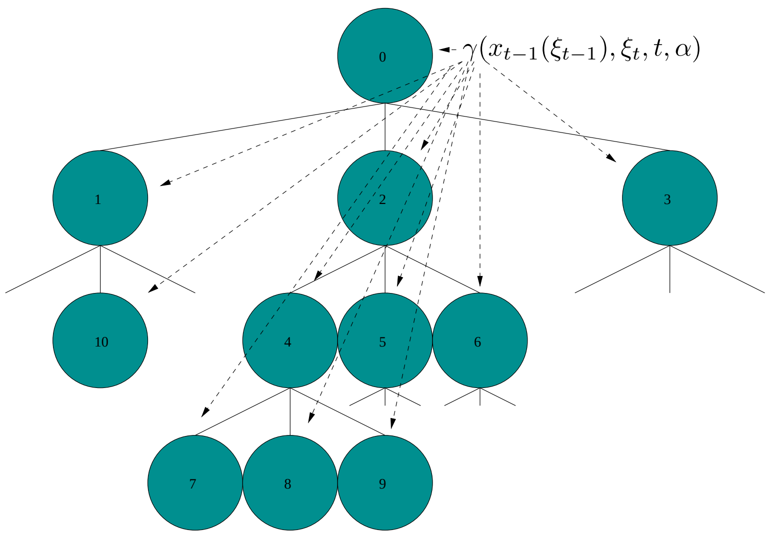

Figure 2: Scenario tree marking random information affecting node 4, parameterized policy.

node. To conclude, in contrast to the stochastic programming approach, which uses only local information, the information used by parameterized policies is global.

The important consequence of the whole scenario tree influencing the decision for a node when using parameterized policies, is that the solution in a node will be reasonable even if the descendants of the node does not provide an accurate description of all possible random outcomes. This makes it possible to reduce the size of the scenario tree without the solution chasing spurious profits. Furthermore, using a parameterized policy makes it possible to use separate scenarios rooted at a common state of the world (also known as a scenario fan) instead of a scenario tree, a relaxation which may simplify the generation of scenarios.

4.2 Flexibility

The major advantage of a stochastic programming based decision support system is the great flexibility it bestows upon its user. Most kinds of restrictions may be handled in a straightforward way. Such restrictions arise for instance from statutory regulations stating restrictions on how the capital may be invested; such rules range from simple bounds on the fraction invested into different assets (see, for instance, Høyland [17]) to more complex rules, taking into account the correlation between different asset classes (see Hilli, Koivu, Pennanen, and Ranne [16]). In addition, funds may be such large market actors that market liquidity may impose constraints on the maximum amount of assets that may be bought or sold during a limited time period.

Policy based systems have difficulties considering the restrictions mentioned above. If we require the constraints on the actions (equation (3e)) hold for all nodes, this will make a number of parameter values α infeasible. In effect this induces a feasible set for the values of α, defined as the set of possible parameter values which do not cause infeasibility in any node. As a consequence of this, we might be prohibited from choosing a set of parameter values giving a good expected objective function value simply because this set of values will cause infeasibility at a single node. This means that the node with potential infeasibility will influence the choice of α to a very large degree, something which is not desirable. A simple way of dealing with restrictions on the actions (which is used in this work) is to soften the policy function by letting it yield a *desired* action $\tilde{y}_t$, instead of the action itself, and then project this desired action onto the closest point in the set of feasible actions $Y_t(x_{t-1}(\xi_{t-1}), \xi_t)$, using a suitable norm.

4.3 Computational price

In order to obtain a good solution from a stochastic programming model, we must construct a scenario tree which gives a sufficiently accurate description of the possible outcomes of the underlying random variables. This means that each node in the scenario tree must have several descendant nodes, making the problem size exponential in the number of time-stages. This ultimately means that the computational requirements of stochastic linear programming problems rapidly become prohibitive as the number of assets and time-stages increases. With a parameterized policy, the number of scenarios need not be increased with the number of time-stages, as the optimization of the policy utilizes information from the whole tree. Then, the trees does not have to branch at each time-stage. Hence, a parameterized policy will scale better with an increasing number of stages.

Solving a stochastic programming problem to get an optimal solution is possible for fairly large problems; see for instance Gondzio and Kouwenberg [14]. However, if the performance and robustness of the model is to be tested, against either historical or simulated data, this requires solving a large number of problem instances which may become prohibitively time consuming. Parameterized policies do not suffer from these drawbacks, as the simulation of the use of such a strategy does not involve any optimization at all; this makes simulation using policies computationally cheap.

4.4 Scenario generation

As have been noted by Klaassen [20], Kouwenberg [21] and Higle [15], the quality of the solution of a stochastic programming problem, measured by how well the solutions perform when applied to out-of-sample data, is highly dependent on the quality of the scenario tree generation. The simplest technique for producing a scenario tree, conditional sampling, has the drawback of producing errors in statistical quantities such as means and covariances. As was mentioned in Section 1.2 several techniques to produce scenario trees of better quality exist. Some of these methods, such as optimizing the scenario tree to fit the statistical parameters of the distribution, require parameters such as means and covariances to be readily available. Furthermore, in complex economy models, such as the system described by Mulvey in [22], these quantities might not be explicitly known. In contrast, since a policy based system uses information from the whole scenario tree when determining the policy, such a system does not need a scenario tree; a set of independent scenarios will do. Hence a policy based system is easier to use in conjunction with a more complex scenario generation mechanism.

4.5 Communication

It is generally not the person who implements an ALM model for a company (or pension fund) who decides on the company's course of action. If the construction of an ALM model is not to remain a theoretical exercise, the results of the modeling must be communicated in an understandable fashion. Since a stochastic programming based model tends to be a black-box solution, it might be hard to convince the people in charge to trust such a model. A parameterized policy has the advantage of being much more transparent, as the decision rule is explicitly available. Even though the funds' manager may not understand the inner workings of the optimization of the policy, the results are clearly available and may be scrutinized, as well as stress tested using different scenarios.

5 A quantitative comparison

5.1 A simple ALM problem

We intend to test whether combining stochastic linear programming and parameterized policy optimization may show a better performance than either of the methods used separately. Therefore we construct a simple (fictional) ALM problem to be used as an illustration. We picture a retirement fund for a corporation, where the sponsor of the fund sets aside means which are to be used to cover future liabilities. The total assets of the fund must also cover a reserve in order to satisfy the rules set by the regulating authorities. Means are allocated to the fund as a fraction of the reserve requirement, and since the corporation sponsoring the fund does not like sudden changes, there are lower and upper bounds on changes of the contribution rate, as well as lower and upper bounds on the contribution rate itself. The lower bound on the contribution rate is negative, which means that the company may actually pull money out of the fund if the investments are successful enough. Since the company is risk-averse, and since failing to cover the reserve is a grave situation, we introduce a set of risk levels, below which the consolidation must not fall, with progressively steeper penalties for lower security margins. Since the company naturally wishes to contribute as little as possible to the fund, we penalize the contribution rate as well. We formally write this problem as

$$
\begin{aligned}
\min \quad & \sum_{t=0}^{T} \tau^t \sum_{q \in Q} \mu_q z_q^t - c^t + \bar{\tau} r^0 \nu + \sum_{t=1}^{t} \tau^{t-1} r^t \nu, && && \text{(4a)}\\
\text{s. t.} \quad & e^0 + \bar{\tau} S^0 r^0 + \sum_{i \in I} (1-\gamma) y_i^{0-} - (1+\gamma) y_i^{0+} = 0, && && \text{(4b)}\\
& e^t + \tau^{t-1} S^t r^t + \sum_{i \in I} (1-\gamma) y_i^{t-} - (1+\gamma) y_i^{t+} = 0, && t = 1, \ldots, T, && \text{(4c)}\\
& x_i^0 = \bar{x}_i \rho_i^0 + y_i^{0+} - y_i^{0-}, && && \text{(4d)}\\
& x_i^t = x_i^{t-1} \rho_i^t + y_i^{t+} - y_i^{t-}, && t = 1, \ldots, T, && \text{(4e)}\\
& c^0 = \frac{\sum_{i \in I} \bar{x}_i \rho_i^0}{S^0} + \bar{\tau} r^0, && && \text{(4f)}\\
& c^t = \frac{\sum_{i \in I} x_i^{t-1} \rho_i^t}{S^t} + \tau^{t-1} r^t, && t = 1, \ldots, T, && \text{(4g)}\\
& z_q^t + c^t \geq f_q, && t = 0, \ldots, T, \\
& && q \in Q, && \text{(4h)}\\
& l_{\Delta r} \bar{\tau} \leq r^0 - \bar{r} \leq u_{\Delta r} \bar{\tau}, && && \text{(4i)}\\
& l_{\Delta r} \tau^{t-1} \leq r^t - r^{t-1} \leq u_{\Delta r} \tau^{t-1}, && t = 1, \ldots, T, && \text{(4j)}\\
& \bar{l} \leq r^t \leq \bar{u}, && t = 0, \ldots, T && \text{(4k)}
\end{aligned}
$$

where variables and parameters are defined by

T,	the last time stage,
I,	the set of investment classes,
Q,	the set of penalties,
x_i^t,	the assets owned of investment class i at time t,
ρ_i^t,	the price development of assets in the period leading up to time t,,
$\bar{x}_i$,	the initial amount owned of asset class i,
y_i^{t+},	the amount of asset class i bought at time t,
y_i^{t-},	the amount of asset class i sold at time t,
γ_i,	the transaction cost for trading with asset i,
r^t,	the contribution rate at time t,
τ^t,	the length of time period t,
$\bar{\tau}$,	the length of the period preceding the first period t,
c^t,	the consolidation (assets over liabilities) at time t,
S^t,	the reserve requirements at time t,
e^t,	the in- or outflow of money at time t (mainly pension payments),
z_q^t,	the level of the violation of cover rule q at time t,
f_q,	the security factor for reserve cover rule q,
μ_q,	the penalty for violating reserve cover rule q,

ν,	the penalty for the contribution rate,
$l_{\Delta r}/u_{\Delta r}$,	the lower/upper bound on the change in contribution rate,
l_r/u_r,	the lower/upper bound on the contribution rate.

As may be seen from the equations (4b)–(4c), the contribution rate is scaled with the length of the period preceding the decision. In essence this means that the company gets to set the contribution rate retroactively. The problem is formulated in this way since we believe that this is a better approximation to a continuously changing contribution than that of requiring the contribution rate to be fixed over a time period, especially when the periods in question are long. In the tests made in this work, seven assets and one reserve are used. All assets and the reserve are assumed to have a joint log-normal distribution. The means, the variances and the correlations of these assets are given in Appendix B together with the other parameters used for our simulations.

5.2 Rolling horizon simulations

Since a comparison or the objective function values from different solution methods will not give any information on how well the methods perform when applied to out of sample scenarios we employ rolling horizon simulations, similar to the ones performed by Kouwenberg [21], Fleten, Høyland and Wallace [12], and Golub et al [13]. The simulations are carried out over a number of test scenarios generated using our model of the reserve and asset returns. For each of these test scenarios we employ the following procedure:

Step 0: Set $t = 0$ and go to Step 2.

Step 1: Use the test scenario and the state of the company just after the decision at time $t-1$ to generate the state of the company just before a decision is made at time t .

Step 2: Use the state of the world of the current sample-path at time t to generate a scenario tree. The size of the tree is adapted so that is lasts to the end of the scenario simulated.

Step 3: Optimize the model over the tree, generating a decision for the company at time t. This decision is used to determine the state of the company after the decision is made at time t, and this state is stored.

Step 4: If $t < T$ then set $t := t + 1$ and go to Step 1.

Step 5: Use the stored states of the company to determine the total penalties and the terminal value of the company; these values are used to evaluate the success of a solution method applied to the current scenario.

The value of a scenario is defined by the total asset value at the end of the simulation, from which we deduct all the penalties incurred on the way.

In step 2 above, we stated that the length of the scenario tree is adapted so that it last to the end of the simulation. If there are a high number of periods remaining, this means that we need to aggregate time-stages in order to get a tree of manageable size. Since a new decision is taken each year in the simulation, we make sure that the first period in the scenario tree is always exactly one year, making the distances to the second decision equal in the simulation and the tree. As the time remaining in the simulations becomes shorter the stages in the scenario trees generated are shortened until each stage is one year. When the remaining time is so short that we have to remove stages, we increase the number of branches at the first stage of the tree in order to keep the number of scenarios in the tree constant (except the two stage tree which has fewer branches). The sizes and stage aggregations of the trees used are given in Table 1 and the aggregation is illustrated in Figure 3.

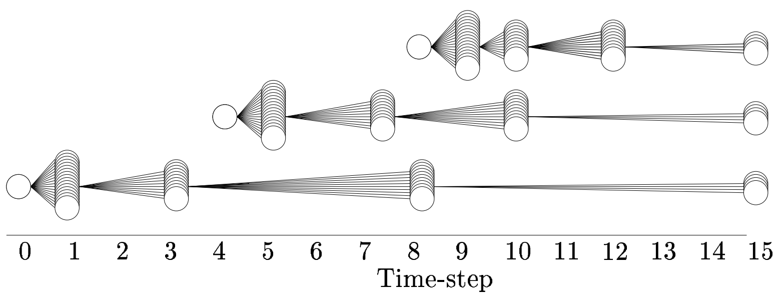

Figure 3: Adaption of the tree size to the remaining scenario length.

The test procedure for the policy based method is similar. However, the policy is optimized only once and the resulting policy is used unaltered throughout all test scenarios. Hence steps 2 and 3 above are replaced by using the policy to determine the state of the company after a decision has been made. Our tests are similar to both those done by Kouwenberg [21] and Fleten, Høyland and Wallace [12], who both use rolling horizon simulations, in which the lengths of the scenario trees are adapted to the remaining lengths of the test scenarios. The main difference is that both Kouwenberg and Høyland et al. use shorter scenarios, which allow them to have the same number of stages in their scenario trees as in the test scenarios, whereas we are forced to aggregate time-stages in the scenario trees.

5.3 Test cases

In our tests we have used two sets of test scenarios, one that is 15 years long, and one that is 10 years long. As the computer resources required to perform rolling horizon simulations increase with the length of the test scenarios, we use 1000 test scenarios for the 10 year set, and 500 test scenarios for the 15 year set in order to get reasonable run-times. Using the chosen sizes of scenario trees, a 5-stage tree results in a problem having approximately 83,000 variables and 46,000 constraints. Each problem takes approximately 8 seconds to solve on a 900Mhz Sun SPARCIII. As each scenario requires the solution of one problem per time-stage (although time-stages with fewer stages takes less time) each 15-period scenario requires approximately slightly less than 2 minutes, making is possible to solve one test case over night.

When a policy-based solution approach is optimized, we use a set of 5000 randomly generated scenarios. The same set of 5000 scenarios is used for all policies, making the comparison easier.

Since we do not know how the assets or the contribution rate should be set in the first stage of the test scenario, we set all transaction costs to 0 and remove the constraint (4i) for the first stage of all simulations.

5.4 Policies used

In this work we test a number of different policies against each other. This section gives a short description of their specifications.

Stochastic programming Naturally, we may address the given test problem using ordinary stochastic linear programming. In this work we have used two shapes of the tree, having four and five stages, respectively. We use different shapes, as we are afraid that the number of branches in the longer tree might be a bit too low. As mentioned previously, the number of stages aggregated in the scenario trees will vary with the remaining length of the test scenarios, as given in Table 1 and Figure 3. The scenario trees are generated by fitting the lower order moments. For 40 and 16 branches, the four lowest moments and the covariances are fitted. For 10 branches, the three lowest moments and the covariances are fitted, and for 4 branches only the two lowest moments are fitted. For higher number of branches, several sets of outcomes are combined. As the random distributions are assumed to be stable, we build a pool of 1000 sets of outcomes for each combination of remaining length and number of branches. Sets of outcomes from these pools are then randomly combined to form trees.

Pure (Bi-linear) parameterized policies The decision which we are to make will essentially vary with two parameters: the current consolidation and the time remaining to the end of the simulation. In order to capture this dependency in the simplest possible way, we construct a bilinear policy. By bilinear we mean that the asset mix chosen is linear in both the consolidation level and the remaining time. The asset mix part of the policy function hence consists of four asset mixes. Each of the four policies represents a corner of the area to be interpolated. The corners are defined as 15 years remaining and no years remaining, respectively, each with one mix for high consolidation and one for low consolidation, respectively. What is considered to be high and low consolidation values are parameters which are optimized. If the consolidation falls outside the interpolation interval, the mix corresponding to the closes extreme value is used.

	stage, long tree				stage, short tree		
period	1	2	3	4	1	2	3
1	640/1	-	-	-	640/1	-	-
2	640/1	10/1	-	-	640/1	10/1	-
3	64/1	10/1	10/1	-	40/1	16/1	10/1
4	16/1	10/1	10/1	4/1	40/1	16/1	10/2
5	16/1	10/1	10/1	4/2	40/1	16/2	10/2
6	16/1	10/1	10/2	4/2	40/1	16/2	10/3
7	16/1	10/1	10/2	4/3	40/1	16/2	10/4
8	16/1	10/2	10/2	4/3	40/1	16/2	10/5
9	16/1	10/2	10/2	4/4	40/1	16/3	10/5
10	16/1	10/2	10/3	4/4	40/1	16/3	10/6
11	16/1	10/2	10/3	4/5	40/1	16/4	10/6
12	16/1	10/2	10/3	4/6	40/1	16/4	10/7
13	16/1	10/2	10/4	4/6	40/1	16/5	10/7
14	16/1	10/2	10/4	4/7	40/1	16/5	10/8
15	16/1	10/2	10/5	4/7	40/1	16/6	10/8

Table 1: Split/aggregated stages for scenario trees.

The desired contribution rate is also specified as a linear function of both the consolidation and the remaining time. Four parameters specify at which consolidation values the contribution rate should be set to its maximum and minimum value for 0 and 15 years remaining, respectively. As constraint (4j) may make it impossible to reach the desired value, we choose the contribution rate which is closest to the desired rate, while still being inside the interval. This corresponds to projecting the desired solution onto the feasible set, as mentioned in Section 4.2.

In order to avoid excessive trading, the asset mix is not forced to exactly comply with the portfolio found by interpolation. Instead, we define a no trade region around the interpolated asset mix, and the final parameter specifies the size of the no-trade region. Hence, the action to take is found as follows:

1 Find the consolidation prior to contribution by dividing the assets held by the reserve. Use this consolidation to find a contribution rate.

2 Project the contribution rate obtained onto the upper and lower bounds defined in equations (4i)–(4j).

3 Use the projected contribution rate to obtain the consolidation after contribution. Use this consolidation to obtain the desired asset mix.

4 Project the current assets held onto the no trade region (centered around the desired asset mix found by interpolation) by minimizing the transaction costs, while requiring the resulting asset mix to be inside the no-trade region.

Table-based policy In order to create a policy based on stochastic programming, we solve the SP-problem for a number of different states of the world, and then find a solution by interpolating between these pre-optimized states. (As we already have a set of solutions ready from the stochastic programming approach, we use them.) In order to perform the interpolation, we form two tables. The first table gives the desired contribution rate as a function of the consolidation *before* the contribution rate income is added. The second one gives the desired asset mix as a function of the consolidation *after* the contribution rate income is added. The tables are formed by sorting the tables from the SLP case by increasing consolidation, after which the solutions are averaged by moving a window of length 20 along the list. In order to capture the solutions dependency on the remaining time, we form one table for each possible remaining scenario length.

We need to be careful when constructing the contribution rate table. As the contribution rate is affected by constraint (4i), we can not use the solutions from the stochastic programming test directly. Instead we obtain new solutions by re-running the entire stochastic programming test case while relaxing the constraint (4i) for all stages of the simulation. (Previously this constraint was relaxed only for the first stage.)

Hybrid policies The hybrid policies are a combination of the table-based policies, and parameterized policies. In these policies, a suggested asset mix and contribution rate are obtained by interpolation from a table, as in the previous policy (the table lookup is based on the consolidation rate). As we believe that the bias from using too few stages in the stochastic programming model will express itself as a tendency of the model to be either too conservative or too risk-taking, we add disturbances to the consolidation used to look up the asset mix and the contribution rate. In essence this means that we trick the model into believing that it is either better off (by increasing the consolidation) or worse off (by decreasing the consolidation).

As in the bilinear policy, we define a no-trade region around the desired asset mix, and a third parameter specifies the width of the no-trade region. The decision at a node is determined in a cascading fashion similar to the case of a pure bilinear policy. First, the consolidation before adding the contribution rate is used to find a desired contribution rate. This desired contribution rate is then projected onto the bounds, giving the consolidation after contributions, which is used to find a desired asset mix. Finally the asset mix within the no-trade region giving the lowest transaction cost is found and used.

An example of how the asset mix table and the no trade region may look for one asset is given in Figure 4. In this figure the dots represent solutions from the pure stochastic programming problem, the dotted lines mark the no-trade region and the whole line is the desired asset fraction invested in asset 4. The line may look erratic, but the variations are rather small. In preliminary tests, we have tried to smooth the curves by manual interpolation, but these alterations gave no effect on the performance of the policy.

The purpose of constructing a hybrid policy is to remove some of the bias stemming from the aggregation of stages in the SLP approach. Since there is no, or little, aggregation at the end of the test scenarios, we will only apply the disturbance for the first eight years of each simulated scenario in 15-year scenarios, and for the first 4 years in in the 10 year scenarios.

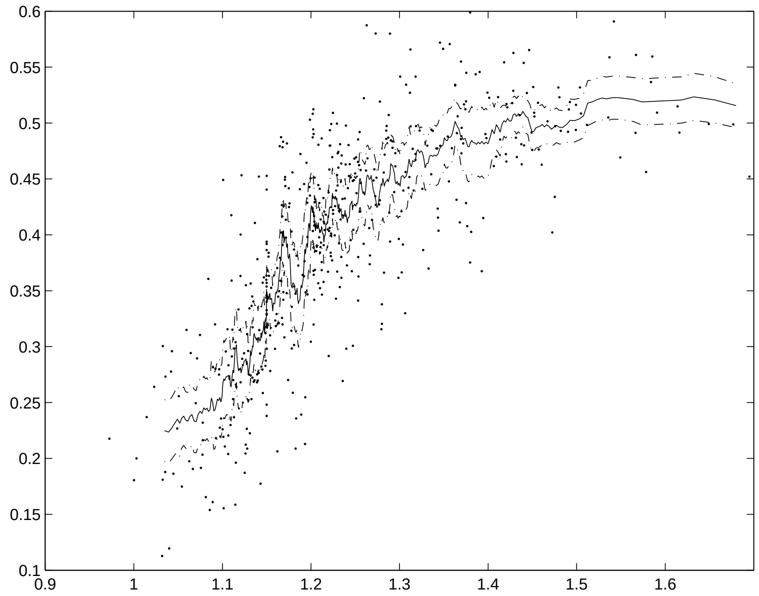

Figure 4: Policy table with no trade region for asset 4, with ten of fifteen years remaining. The dots are individual SLP solutions.

5.5 Numerical results

The results from our numerical experiments on 15 year scenarios are given in Tables 2 and 3. As might be seen from these tables, the difference between the results from using a map based on stochastic linear programming, and the SLP itself is not significant. This either means that we do not suffer any ill effects from using tables in this manner, or that we have two effects cancelling each other out. The table based approach should have an advantage over the SLP-based approach, as several solutions are averaged, decreasing the SLP-based solutions' tendency to chase spurious profits arising from the scenario trees inability to describe the underlying true random distribution. On the other hand, the table-based policy ignores the assets held before trade, which makes this approach incur higher transaction costs.

There is however a difference in performance between the hybrid policies and the SLP-based approach, and this difference is larger for the three-period scenario trees. This is consistent with our hypothesis that using a hybrid policy will remove

Policy	Mean cons.	Mean pen.	Mean tot.	Cost increase
Bilinear policy	1.2081	3.2810	-2.0729	3.12±0.78%
SLP long tree	1.2274	3.2617	-2.0344	1.73±0.57%
SLP short tree	1.2141	3.3356	-2.1216	4.88±0.85%
Map, long tree	1.2273	3.2586	-2.0312	1.62±0.43%
Map, short tree	1.2122	3.3323	-2.1200	4.82±0.82%
Hybrid, long tree	1.2581	3.2446	-1.9865	0%
Hybrid, short tree	1.2499	3.2733	-2.0234	1.33±0.37%

Table 2: Results for different policies, 15 year scenarios. Cost increase is objective function difference recalculated to increase in contributions, compared to the best case (with standard deviation).

	SLP, long tree	Map, long tree
Map, long tree	0.78	-
Hybrid, long tree	0.0025	0.00017

Table 3: Statistical strength of difference between selected methods (probability of obtaining larger absolute deviation from 0 given that no difference exists).

Policy	Mean cons.	Mean pen.	Mean tot.	Cost increase
SLP long tree	1.2059	2.4114	-1.2055	0.48±0.50%
Map, long tree	1.2028	2.4101	-1.2073	0.57±0.23%
Hybrid, long tree	1.2095	2.4054	-1.1959	

Table 4: Results for different policies, 10 year scenarios.

some bias from using aggregated time periods, as the three-period scenario trees aggregate more stages, and hence should suffer from more bias.

The optimal parameter setting for the hybrid policy is $[-0.032, 0.096, 0.47]$ (for the long tree case). This means that when the policy looks up the contribution rate, it lowers the consolidation, resulting in a higher contribution rate. When the policy looks up the asset mix, it increases the consolidation, resulting in a riskier asset mix. In Figure 5 we plot the average consolidation, the average contribution rate and the average yield of the asset portfolio for four different approaches. As may be seen from this figure, the hybrid policy alters the solutions from the SLP approach by setting a higher contribution rate at the beginning of the scenario, while investing in a riskier portfolio. It seems as if the SLP approach chooses a too conservative portfolio, while the SLP solution using short trees is more conservative still.

Looking at the results for the 10 year test case, available in Table 4, we still see a difference between the hybrid policy and the pure SLP solutions, however not as large as for the 15-year scenarios. Using a pairwise T-test, we see that the probability of obtaining a larger difference given that none exists, is 0.059, making the results less clear than for the 15 year test case. Having a smaller difference for the 10 year case is consistent with our theory that we remove some bias from the SLP solutions by using hybrid policies, since the short case will be less aggressively aggregated, and hence these solutions will contain less bias. If we look at the optimal solution for the 10 year test case, the control parameters have the values $[-0.0055, 0.046, 0.37]$. Hence the optimal offset for the 10 year case is smaller that for the 15 year case, again indicating that the shorter test case has a smaller bias.

Well worth noting is that the policy optimization problems are not generally convex (although this of course depends on the policy and the problem considered). The value of using a policy on a set of scenarios is a noisy non-convex, non-differentiable function of the policy parameters. The reason underlying the non-differentiability is the discrete nature of both the policy and the penalties used. In order to explain this, we consider the case when the parameterized policy is applied to one scenario. As the penalties for low consolidation are applied in steps, there exists a set of policy parameters such that the consolidation for the last stage has exactly the value of one such step. Hence the value of using the policy will be non-differentiable at this set of parameters, as an increase and a decrease in consolidation will carry different rewards. The use of a no-trade region

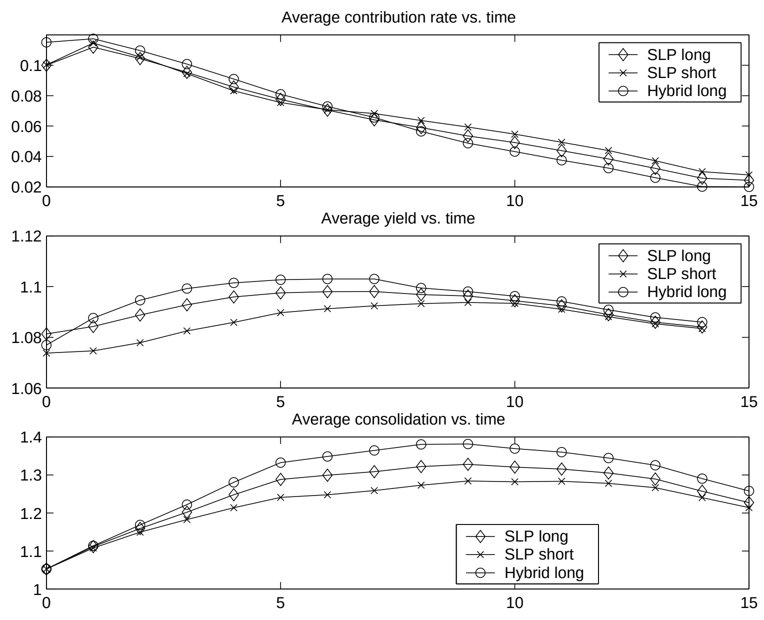

Figure 5: Development of three different strategies.

will result in the same phenomenon: there exist sets of parameter values such that the policy will be on the boundary of the no-trade region, possibly resulting in a non-differentiability. When we optimize the policy function, we use a large number of scenarios, each of which will have a number of non-differentiable points, making the policy function consist of a number of continuous facets. As the number of scenarios used for evaluation increases, the number of facets increases, making the scale of the ruggedness of the function surface smaller. Furthermore, if the policy function would be applied to a continuous random distribution, the surface would be smooth.

In order to illustrate the non-convexity and non-differentiability of the policy problem, we evaluate the hybrid policy for 5000 scenarios of length 15 years, while varying the two first policy parameters. The result from these test runs are plotted in Figure 6, illustrating the non-convexity of the problem. As a consequence of this non-convexity, we can not be sure that we have actually found the optimum when we are searching for the optimal set of parameters in the hybrid and bilinear policies. In order to decrease the risk of getting stuck in a bad local minimum, we start the optimization by performing a grid-search for the hybrid policies. As the bi-linear policy has 37 parameters, an accurate grid search is not possible. For this strategy we instead try to avoid local minima by starting from a large number of random starting points. These random points are centered around an approximate solution, found by manually fitting the policy parameters to make the policy function match the solutions obtained from the stochastic programming approach.

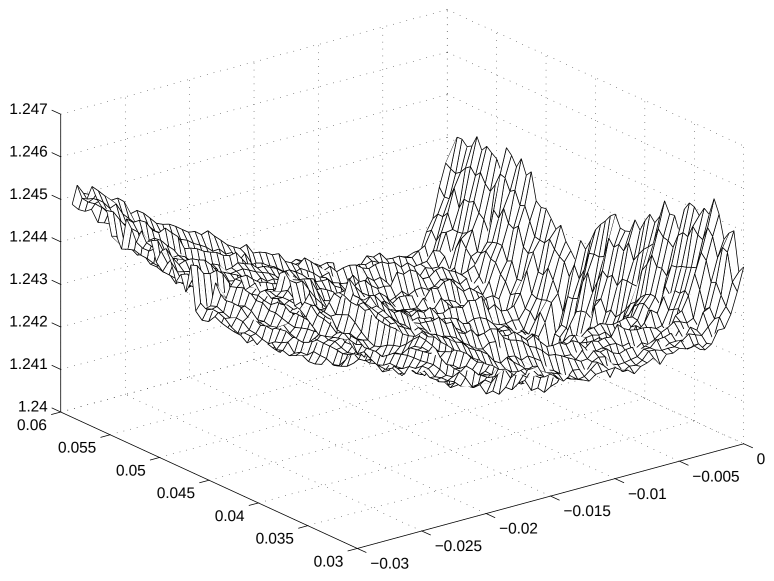

Figure 6: Hybrid policy value as a function of the two consolidation disturbances.

6 Conclusions and further work

In this work we have made a comparison between parameterized policies and stochastic linear programming for ALM purposes. We have shown that tabulating solutions from a number of stochastic linear programming (SLP) solutions may speed up the running of test cases without a significant loss in performance. In addition we have shown that using table-based policies in combination with optimized disturbances improves the performance for long scenarios, as we may remove some bias stemming from the aggregation of stages in the SLP approach. Apart from the increases in performance, a policy based ALM system is easier to interpret and communicate as the policies are explicitly available for inspection.

By combining a SLP based system with a policy based system some advantages over the two separate systems may be made. An SLP based system has the advantage of requiring few prior assumptions, as the user of the system gives no input on the properties of the solution. The main disadvantage of an SLP based system is its inability to handle a large number of time periods and/or a large number of assets. In addition, an SLP based system is impractical to work with for testing, as the computational requirements are rather high.

As for pure parameterized policies, their main advantage over an SLP based system is their ability to handle a large number of time-periods. In addition, a policy based system does not need a scenario tree: a set of independent scenarios is sufficient, which makes it possible to use parameterized policies in conjunction with complex scenario generation methods.

There are two major drawbacks of using a policy based system: firstly, the user must still specify the policy function, which is a non-trivial task. Secondly, optimizing policy functions are generally non-convex problems which have an objective function which is expensive to evaluate, making them hard to solve, even though the number of variables is small.

When we combine the two systems, we will reduce the problems of simulating the solutions of a stochastic programming based approach, as well as make the results easier to interpret, without sacrificing performance. In addition, the problems of selecting the shape of the policy function is reduced, although we still need to choose the shape of the disturbance. (Choosing a bad shape for the disturbance should however not have as great an impact on the overall result as a misspecified pure policy function.) The major drawback of using a hybrid approach is the fact that we need to implement both an SLP based solution, and a policy based one. Hence the hybrid policies may more appropriately be seen as a heuristic to improve an SLP-based model.

In this work we have only looked at one specific and extremely simple instance of an ALM-problem. In addition, there is essentially only one state: the consolidation. The purpose of this work has been to explore whether this method has any potential of increasing the performance of our model of a Swedish life insurance company. As the the results are positive, the next logical step would be to implement hybrid policies for that model. Previously we have only implemented the pure table-based policy with no optimization for this model. Just as in this work we found no significant difference between using a table-based policy and using the SLP-solution directly. For the insurance model we also found that using longer, more narrow trees increased performance, even when the lower number of branches made the solutions very unstable.

Currently the implementation of the hybrid policy is inefficient, as we use SLP solutions from a number of rolling horizon simulations to form our tables. A better approach is to more carefully select the SLP-problems to be solved, to span the interesting region of possible states. This might even be made on demand, solving new SLPs when the database fails to provide a good set of solutions to interpolate from.

References

[1] F. Altenstedt, *Asset aggregation in stochastic programming models for asset liability management*, Preprint 2003:48, Chalmers University of Technology, Department of mathematics, SE-412 96 Göteborg, Sweden, 2003. Submitted to Computational Optimization and Applications.

[2] ——, *An asset liability management system for a Swedish life insurance com-*

pany, Preprint 2003:47, Chalmers University of Technology, Department of mathematics, SE-412 96 Göteborg, Sweden, 2003. Submitted to Annals of Operations Research.

[3] P. Barth and A. Bockmayr, *Modelling mixed-integer optimisation problems in constraint logic programming*, Tech. Rep. MPI-I-95-2-011, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, November 1995.

[4] J. Benders, *Partitioning procedures for solving mixed variables programming problems*, Numeriche Mathematik, 4 (1962), pp. 238–252.

[5] G. C. Boender, *A hybrid simulation/optimisation scenario model for asset/liability management*, European Journal of Operational Research, 99 (1997), pp. 126–135.

[6] D. R. Cariño, D. H. Myers, and W. T. Ziemba, *Concepts, technical issues, and uses of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 450–462.

[7] D. R. Cariño and W. T. Ziemba, *Formulation of the Russell–Yasuda Kasai financial planning model*, Operations Research, 46 (1998), pp. 433–449.

[8] D. R. Cariño, W. T. Ziemba, T. Kent, D. H. Myers, C. Stacey, M. Sylvanus, A. L. Turner, and K. Watanabe, *The Russell–Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming*, Interfaces, 24 (1994), pp. 29–49.

[9] A. Conn, K. Sheinberg, and P. Toint, *recent progress in unconstrained nonlinear optimization without derivatives*, Mathematical Programming, 79 (1997), pp. 397–414.

[10] G. Consigli and M. Dempster, *Dynamic stochastic programming for asset–liability management*, Annals of Operations Research, 81 (1998), pp. 131–161.

[11] C. Dert, *Asset liability management for pension funds; A multistage chance constrained programming approach*, PhD thesis, Erasmus University Rotterdam, 1995.

[12] S.-E. Fleten, K. Høyland, and S. W. Wallace, *The performance of stochastic dynamic and fixed mix portfolio models*, European Journal of Operational Research, 140 (2002), pp. 37–49.

[13] B. Golub, M. Holmer, R. McKendall, L. Pohlman, and S. Zenios, *A stochastic programming model for money management*, European Journal of Operational Research, 85 (1995), pp. 282–296.

[14] J. Gondzio and R. Kouwenberg, *High performance computing for asset liability management*, Operations Research, 49 (2001), pp. 879–891.

[15] J. L. Higle, *Variance reduction and objective function evaluation in stochastic linear programs*, INFORMS Journal on Computing, 10 (1998), pp. 236–247.

[16] P. Hilli, M. Koivu, T. Pennanen, and A. Ranne, *A stochastic programming model for asset liability management of a Finnish pension company*, Stochastic programming E-print series, (2003). http://dochost.rz.hu-berlin.de/speps/.

[17] K. Høyland, *Asset liability management for a life insurance company: A stochastic programming approach*, PhD thesis, Department of Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway, 1998.

[18] K. Høyland and S. W. Wallace, *Analyzing legal regulations in the Norwegian life insurance business using a multistage asset-liability management model*, European Journal of Operational Research, 134 (2001), pp. 293–308.

[19] ——, *Generating scenario trees for multistage decision problems*, Management Science, 47 (2001), pp. 295–307.

[20] P. Klaassen, *Discretized reality and spurious profits in stochastic programming models for asset liability management*, European Journal of Operational Research, 101 (1997), pp. 374–392.

[21] R. Kouwenberg, *Scenario generation and stochastic programming models for asset liability management*, European Journal of Operational Research, 134 (2001), pp. 279–292.

[22] J. M. Mulvey, *Generating scenarios for the Towers Perrin investment system*, Interfaces, 26 (1996), pp. 1–15.

[23] J. M. Mulvey, G. Gould, and C. Morgan, *An asset and liability management system for Towers Perrin–Tillinghast*, Interfaces, 30 (2000), pp. 96–114.

[24] A. F. Perold and W. F. Sharpe, *Dynamic strategies for asset allocation*, Financial Analysts Journal, (1988), pp. 16–27.

[25] G. Pflug, *Scenario tree generation for multiperiod financial optimization by optimal discretisation*, Mathematical Programming, Series B, 89 (2001), pp. 251–271.

[26] R. M. Van Slyke and R. Wets, *L-shaped linear programs with applications to optimal control and stochastic programming*, SIAM Journal on Applied Mathematics, 17 (1969), pp. 638–663.

A Implementation details

All models in this work were written in STOCHPLAM (available from www.math.chalmers.se/∼alten/stoplam/), a stochastic programming extension by the author of the open source algebraic modeling language PLAM by Barth and Bockmayr [3]. The solver used for stochastic linear programming is the BNBS solver using CPLEX as an underlying LP-solver (BNBS is available via the web-page www.stoprog.org). The policies are optimized using the program package DFO (Derivative Free Optimization, available from www.coin-or.org), by Conn, Toint and Scheinberg [9], which is a surrogate optimization algorithm. A model of the function is built using function evaluations, after which set of parameters minimizing the model (inside a trust region) is found, and the real function is evaluated for this point.

B Simulation data

Parameter	value
Q	{1,2,3,4}
$\sum_{i\in I}\bar{x}_i$	1.0
γ_i	0.005, $i \in I$
S^0	0.95
e^t	0, $t = 0, \ldots, T$
f_1	1.00
f_2	1.04
f_3	1.08
f_4	1.15
μ_1	6
μ_2	4
μ_3	2
μ_4	1
ν	3
$l_{\Delta r}$	-0.02
$u_{\Delta r}$	0.02
l_r	-0.04
u_r	0.12

Table 5: Parameters used in simulations.

	reserve	a1	a2	a3	a4	a5	a6	a7
mean	13.71	6.59	7.37	11.95	10.84	4.59	8.19	5.81
Std.	1.93	5.28	9.46	24.71	18.00	0.43	16.06	3.50

Table 6: Mean and standard deviation of assets used (%).

	reserve	a1	a2	a3	a4	a5	a6
a1	0.49381						
a2	0.18627	0.18943					
a3	0.45105	0.45642	0.20575				
a4	0.33364	0.33703	0.58166	0.67816			
a5	0.56793	0.61823	0.11801	0.26933	0.19448		
a6	0.48187	0.48950	0.03016	0.15257	0.01430	0.29368	
a7	0.19084	0.19593	0.17999	0.21120	0.09766	0.16223	0.17333

Table 7: Correlations of assets used.